

MALLOW 2009

M. Baldoni, C. Baroglio, J. Bentahar, G. Boella, M. Cossentino, M. Dastani, B. Dunin-Kępicz, G. Fortino, M.-P. Gleizes, J. Leite, V. Mascardi, J. Padget, J. Pavón, A. Polleres, A. El Fallah Seghrouchni, P. Torroni, R. Verbrugge (eds.)

Multi-Agent Logics, Languages, and Organisations

*Second International Federated Workshops,
MALLOW'009, Workshops proceedings*

Turin, Italy, September 7-10, 2009

MALLOW'009 Home Page:
<http://agents009.di.unito.it/MALLOW.html>

Preface

The Multi-Agent Logics, Languages, and Organisations Federated Workshops (MALLOW for short), in its *second edition* this year after the success of MALLOW'007 held in Durham (UK), is a forum for researchers interested in sharing their experiences in agents and multi-agent systems. MALLOW'009 was held at the Educatorio della Provvidenza, in Torino (Italy), from September 7th, 2009 through September 10th, 2009.

In particular, this edition includes the workshops:

- Agents, Web Services and Ontologies, Integrated Methodologies (MALLOW-AWESOME'009);
- Coordination, Organization, Institutions and Norms in Agent Systems & On-line Communities (COIN@MALLOW'009);
- Formal Approaches to Multi-Agent Systems (FAMAS'09);
- Languages, methodologies and Development tools for multi-agent systems (LADS'009);
- Multi-Agent Systems and Simulation (MAS&S'09).

MALLOW-AWESOME'009 wants to stimulate discussion among researchers working on Agents, Web Services, and Ontologies, in order to help the identification and the definition of Methodologies for integrating them. The realisation of distributed, open, dynamic, and heterogeneous software systems is, in fact, a challenge that involves many facets, from formal theories to software engineering and practical applications. Scientists in various research areas, such as Semantic Web, Web Services, Agents, Ontologies, are attacking this problem from different perspectives. MALLOW-AWESOME'009 attempts to provide a discussion forum for collecting and comparing such diverse experiences with the aim of fostering cross fertilization.

COIN@MALLOW'009 belongs to the COIN workshop series, which brings together the topics of coordination, organization, institutions and norms in the context of multi-agent systems. This edition of COIN focusses on these issues in the context of on-line communities, where we seek contributions that explore the dimensions of social, legal, economic and technological norms as they affect agent-agent, agent-human, human-human interactions.

FAMAS'09, the fourth edition of the FAMAS workshop series, after FAMAS'03 affiliated to ETAPS'03 in Warsaw, FAMAS'06 affiliated with ECAI'06 in Riva del Garda, and FAMAS'007 affiliated with MALLOW'007 in Durham, aims at bringing together researchers from the fields of logic, theoretical computer science and multiagent systems in order to discuss formal techniques for specifying and verifying multi-agent systems.

LADS'009 aims to offer a rich forum for leading researchers, from both academia and industry, interested in sharing their experiences about the theory

and practice of formal approaches, programming languages, tools and techniques that support the development and deployment of multi-agent systems. These are gaining increasing attention in important application areas such as electronic institutions, semantic web, web services, security, grid computing, ambient intelligence, pervasive computing, electronic contracting, among others.

MAS&S'09 aims at providing a forum for discussing recent advances on the integration of Simulation and Agent Oriented Software Engineering (AOSE) methodologies and techniques for the analysis, design, validation and implementation of Multi-Agent Systems.

MALLOW'009 has hosted also a special event, the COST AT Argumentation Day, organized by Guido Boella and Leendert van der Torre. The event discussed new directions in argumentation, such as temporal dynamics, fibring of argumentation frameworks, modal logics of argumentation, modal provability foundations for argumentation, etc. The day was opened by one of the four MALLOW'009 invited speakers, *Dov Gabbay*.

Besides Dov Gabbay, the organizers were very honored to have also *Fabio Bellifemine*, *Alexis Tsoukias*, and *Franco Zambonelli* as invited speakers.

Following the tradition of the previous edition, MALLOW'009 was part of Agents'009 that includes also EASSS'09, the eleventh edition of the *European Agent Systems Summer School*.

This volume contains the proceedings of the five workshops, for a total of forty-seven high quality papers, which were selected by the programme committees of the workshops for presentation. The volume is organized as follows. Besides this overall presentation, each workshop has an introductory essay, authored by the organizers, which presents the workshop, lists the programme committee members and additional reviewers, the accepted papers with their authors, and the workshop sponsors. It is followed by the workshop papers. The table of contents of this volume reports, for each workshop, the page number of its introductory essay and the first page of the workshop papers.

We would like to thank all authors for their contributions, the members of the Steering Committee for the precious suggestions and support, and the members of the Programme Committees and the additional reviewers for the excellent work during the reviewing phase, the sponsors, the head of the Eudatorio della Provvidenza, and all the persons who helped the organization of this event.

August 22th, 2009

Matteo Baldoni
Cristina Baroglio
Jamal Bentahar
Guido Boella

Massimo Cossentino
Mehdi Dastani
Barbara Dunin-Kępicz
Giancarlo Fortino
Marie-Pierre Gleizes
João Leite
Viviana Mascardi
Julian Padget
Juan Pavón
Axel Polleres
Amal El Fallah Seghrouchni
Paolo Torroni
Rineke Verbrugge

Local Organizers

Matteo Baldoni	Università degli Studi di Torino, Italy
Cristina Baroglio	Università degli Studi di Torino, Italy
Guido Boella	Università degli Studi di Torino, Italy

Workshops Organizers

MALLOW-AWESOME'009 Workshop Organizers

Matteo Baldoni	Università degli Studi di Torino, Italy
Cristina Baroglio	Università degli Studi di Torino, Italy
Jamal Bentahar	Concordia University, Canada
Viviana Mascardi	Università degli Studi di Genova, Italy

COIN@MALLOW'009 Workshop Organizers

Julian Padget	University of Bath, UK
Axel Polleres	National University of Ireland, Galway

FAMAS'09 Workshop Organizers

Barbara Dunin-Kępicz	Warsaw University, Poland
Rineke Verbrugge	University of Groningen, The Netherlands

LADS'009 Workshop Organizers

Mehdi Dastani	Utrecht University, The Netherlands
Amal El Fallah Seghrouchni	University of Paris VI, France
João Leite	Universidade Nova de Lisboa, Portugal
Paolo Torroni	University of Bologna, Italy

MAS&S'09 Workshop Organizers

Giancarlo Fortino	Universit della Calabria, Italy
Massimo Cossentino	ICAR/CNR, Italy
Juan Pavón	Universidad Complutense Madrid, Spain
Marie-Pierre Gleizes	IRIT - Université Paul Sabatier, France

Additional local collaborators

Patrizio Barbini	Marco Martin	Claudio Schifanella
Roberto Grenna	Claudio Mattutino	Serena Villata
Elisa Marengo		

Steering Committee

Thomas Agotnes	University of Bergen, Italy
Cristina Baroglio	Università degli Studi di Torino, Italy
Rafael H. Bordini	Federal University of Rio Grande do Sul, Brasil
Virginia Dignum	Utrecht University, The Netherlands
João Leite	Universidade Nova de Lisboa, Portugal
John Lloyd	Australian National University
Pablo Noriega	Consejo Superior de Investigaciones Científicas, Spain
Amal El Fallah Seghrouchni	University of Paris VI, France
Munindar P. Singh	North Carolina State University, USA
Rineke Verbrugge	University of Groningen, The Netherlands

Sponsoring Institutions



Table of Contents

Agents, Web Services and Ontologies, Integrated Methodologies, MALLOW-AWESOME'009 (Introductory Essay of the Workshop)	1
<i>Matteo Baldoni, Cristina Baroglio, Jamal Bentahar, Viviana Mascardi</i>	
MALLOW-AWESOME'009 Papers	5
MALLOW Workshop on Coordination, Organization, Institutions and Norms in Agent Systems & On-line Communities (COIN@MALLOW'009)	73
<i>Julian Padget, Axel Polleres</i>	
COIN@MALLOW'009 Papers	77
Introduction to the Proceedings of FAMAS'09 Formal Approaches to Multiagent Systems	117
<i>Barbara Dunin-Kęplicz, Rineke Verbrugge</i>	
FAMAS'09 Papers	121
LAnguages, methodologies and Development tools for multi-agent systemS, LADS'009	203
<i>Mehdi Dastani, Amal El Fallah Seghrouchni, João Leite, Paolo Torroni</i>	
LADS'009 Papers	207
The 3 rd International Workshop on Multi-Agent Systems and Simulation (MAS&S): Towards an Integration of Agent-Oriented Software Engineering and Simulation, MALLOW-MAS&S'09	293
<i>Giancarlo Fortino, Massimo Cossentino, Juan Pavón, Marie-Pierre Gleizes</i>	
MAS&S'09 Papers	297

Agents, Web Services and Ontologies, Integrated Methodologies, MALLOW-AWESOME'009

(Introductory Essay of the Workshop)

Matteo Baldoni*, Cristina Baroglio*, Jamal Bentahar†, and Viviana Mascardi‡

*Dipartimento di Informatica, Università degli Studi di Torino
c.so Svizzera, 185 — I-10149 Torino (Italy)

Email: {baldoni,baroglio}@di.unito.it

†Concordia Institute for Information Systems Engineering, Concordia University
1515 Ste-Catherine Street West, EV7.630
Montreal, Quebec, H3G 2W1, Canada

Email: bentahar@ciise.concordia.ca

‡Dipartimento di Informatica e Scienze dell'Informazione, Università degli Studi di Genova
Via Dodecaneso, 35, 16146 Genova (Italy)

Email: mascardi@disi.unige.it

Abstract

Following the success of the previous edition held in Durham in 2007, MALLOW-AWESOME'009 wants to stimulate discussion among researchers working on Agents, Web Services, and Ontologies, in order to help the identification and the definition of Methodologies for integrating them. The realisation of distributed, open, dynamic, and heterogeneous software systems is, in fact, a challenge that involves many facets, from formal theories to software engineering and practical applications. Scientists in various research areas, such as Semantic Web, Web Services, Agents, Ontologies, are attacking this problem from different perspectives. MALLOW-AWESOME'009 attempts to provide a discussion forum for collecting and comparing such diverse experiences with the aim of fostering cross fertilization.

I. INTRODUCTION

MALLOW-AWESOME'009 is at its second edition this year, the first edition has been held in Durham, UK [1]. It was born for stimulating discussion among researchers and practitioners working on Agents, Web Services, and Ontologies, in order to help the identification and the definition of Methodologies for integrating them.

The realisation of distributed, open, dynamic, and heterogeneous software systems is, in fact, a challenge that involves many facets, from formal theories to software engineering and practical applications. Scientists in various research areas, such as Semantic Web, Web Services, Agents, Ontologies, are attacking this problem from different perspectives. MALLOW-AWESOME'009 provides a discussion forum for collecting and comparing such diverse experiences with the aim of fostering cross fertilization.

MALLOW-AWESOME'009 was in Torino, Italy, held as part of MALLOW'009, the second edition of Multi-Agent Logics, Languages, and Organisations (Federated Workshops), in Torino, Italy.

This volume contains the six papers and the three extended abstracts that have been selected by the Programme Committee for presentation at the workshop. Each paper received at least three reviews in order to supply the authors with a rich feedback. The contributions submitted to MALLOW-AWESOME'009 cover hot topics in the fields of rule-based declarative representation of services, web service a-priori composition and synchronisation, methodological issues, extensions to SOA/WS systems with agents' features, extensions of agent platforms with workflow management, applications.

Selected and expanded papers will be published as a special issue in a high-quality international journal: negotiations with the editors are still on their way.

We would like to thank all authors for their contributions and the members of the Programme Committee for the excellent work during the reviewing phase.

Matteo Baldoni
Cristina Baroglio
Jamal Bentahar
Viviana Mascardi
August 13, 2009

II. WORKSHOP COMMITTEES

A. Workshop Organizers

Matteo Baldoni	Università degli Studi di Torino, Italy
Cristina Baroglio	Università degli Studi di Torino, Italy
Jamal Bentahar	Concordia University, Canada
Viviana Mascardi	Università degli Studi di Genova, Italy

B. Programme Committee

Federico Bergenti	Università di Parma, Italy
Mario Bravetti	Università di Bologna, Italy
Antonio Brogi	Università di Pisa, Italy
Federico Chesani	Università di Bologna, Italy
Marco Comuzzi	City University London, United Kingdom
Virginia Dignum	Utrecht University, The Netherlands
Amal El Fallah Seghrouchni	University of Paris 6, France
Pilar Herrero	Universidad Politécnica de Madrid, Spain
Benjamin Hirsch	Technische Universitaet Berlin, Germany
Fuyuki Ishikawa	National Institute of Informatics, Japan
Zakaria Maamar	Zayed University, United Arab Emirates
Peter Massuthe	Humboldt University Berlin, Germany
Julian Padget	University of Bath, United Kingdom
Viviana Patti	Università di Torino, Italy
Adam Pease	Articulate Software, California, USA
Axel Polleres	National University of Ireland, Ireland
Alessandro Ricci	Università di Bologna, Italy
Birna van Riemsdijk	Delft University, The Netherlands
Giovanni Rimassa	Whitestein Technologies AG, Zurich, Swiss
Munindar Singh	North Carolina State University, USA
Christopher Walton	Metaforic, United Kingdom
Hamdi Yahyaoui	KFUPM, Saudi Arabia
Nobuko Yoshida	Imperial College London, United Kingdom
Muhammad Younas	Oxford Brookes University, United Kingdom

III. LIST OF PAPERS

- Joint Achievement of Services' Personal Goals
by *Matteo Baldoni, Cristina Baroglio, Elisa Marengo, Viviana Patti, and Claudio Schifanella*
- Verifying A-Priori the Composition of Declarative Specified Services
by *Federico Chesani, Paola Mello, Marco Montali, and Paolo Torroni*
- Web Services Synchronization in Composition Scenarios
by *Hamdi Yahyaoui, Zakaria Maamar, Jamal Bentahar, and Khouloud Boukadi*

- Enhancing Engineering Methodology for Communities of Web Services
by *Mohamed Elmenshawy Mohamed, Jamal Bentahar, and Rachida Dssouli*
- Programming SOA/WS Systems with Cognitive Agents and Artifact-Based Environments
by *Michele Piunti, Andrea Santi, and Alessandro Ricci*
- Exploiting Agents and Ontologies for Type- and Meaning-Safe Adaptation of Java Programs
by *Viviana Mascardi and Davide Ancona*
- The CAWE Framework - Enhancing Service Oriented Architecture with Context Awareness (Extended Abstract)
by *Liliana Ardissoni, Anna Goy, Roberto Furnari, Giovanna Petrone, and Marino Segnan*
- AgentSeeker: an Ontology-based Enterprise Search Engine (Extended Abstract)
by *Andrea Passadore, Alberto Grosso, and Antonio Boccalatte*
- WADE: An Open Source Platform for Workflows and Agents (Extended Abstract)
by *Giovanni Caire, Elena Quarantotto, and Giovanna Sacchi*

IV. SPONSORING INSTITUTIONS

Matteo Baldoni has partially been funded by Regione Piemonte through the project ICT4LAW.

Cristina Baroglio has partially been funded by Regione Piemonte through the project ICT4LAW.

Jamal Bentahar is funded by the Natural Sciences and Engineering Research Council of Canada (NSERC 341422-07: Multi-agent systems for advanced semantic grid computing), by Le Fonds québécois de la recherche sur la nature et les technologies (FQRNT 2008-NC-119348: An advanced model based on argumentation and computational logic for secure agent negotiation), and partially by Fonds de recherche sur la société et la culture (FQRSC 111881: A theory for communication and coordination in artificial intelligence).

Viviana Mascardi has partially been funded by the Iniziativa Software CINI-Finmeccanica project, <http://www.iniziativasoftware.it/>.

REFERENCES

- [1] M. Baldoni, C. Baroglio, and V. Mascardi, editors. *Proceedings of the Multi-Agent Logics, Languages, and Organisations, Federated Workshops, MALLOW'007, Agent, Web Services and Ontologies, Integrated Methodologies (MALLOW-AWESOME'007) workshop*, Durham, GB, September 2007.

Joint achievement of services' personal goals

Matteo Baldoni, Cristina Baroglio, Elisa Marengo, Viviana Patti, Claudio Schifanella

Dipartimento di Informatica — Università degli Studi di Torino

c.so Svizzera, 185, I-10149 Torino, Italy

Email: {baldoni, baroglio, emarengo, patti, schi}@di.unito.it

Abstract—Web service specifications can be quite complex, including various operations and message exchange patterns. In this work, we give a rule-based declarative representation of services, and in particular of WSDL operations, that enables the application of techniques for reasoning about actions and change, that are typical of agent systems. This makes it possible to reason on a rule-based specification of choreography roles and about the selection of possible role players on a goal-driven basis, and allows us to attack the problem of the joint achievement of individual goals for a set of services which animate a choreography.

I. INTRODUCTION

Declarative languages are becoming very important in the Semantic Web, since the focus started to shift from the ontology layer to the *logic layer*, with a consequent need of expressing rules and of applying various forms of reasoning¹, an interest also witnessed by the creation of a W3C working group to define a Rule Interchange Format². Particularly challenging applications concern *web services*.

One of the key ideas behind web services is that services should be amenable to automatic retrieval, thus facilitating their *re-use*. Nevertheless, retrieval cannot yet be accomplished automatically as well and as precisely as desired because the current web service representations (mainly WSDL [1] and BPEL [2]) and the discovery mechanisms are semantically poor and lack of sufficient flexibility. As shown by the framework presented in this paper, *rule-based declarative languages*, and the *reasoning techniques* that they support, supply both an expressive formal semantics and flexibility in the accomplishment of the service selection task.

The need of adding a *semantic layer* to service descriptions is not new. Some approaches, e.g. OWL-S [3] and WSMO [4], propose a richer annotation, aimed at representing *inputs*, *outputs*, *preconditions* and *effects* of the service. Inputs and outputs are usually expressed by ontological terms, while preconditions and effects are often expressed by means of *logic representations*. Preconditions and effects are also used in *design by contract*, originally introduced by Meyer for the EiffelTM language [5]. Here preconditions are the part of the contract which is to be guaranteed by the client; if this condition is guaranteed in the execution context of a method, then the server commits to guaranteeing that the postcondition holds in the state reached by the execution.

On the other hand, there is often the need of using services not in an individual way but jointly, for executing tasks that

none of them alone can accomplish. Semantic annotations alone are not sufficient in this case; it becomes useful to introduce a notion of *goal* [6], [7], [8], which can be used to guide both the selection and the composition of services. The introduction of goals opens the way to the introduction of another abstraction, that of *agent*. Agents include not only the ability of dealing with goals and of performing goal-driven forms of reasoning, but they also show autonomy and proactivity, which are characteristics that help when dealing with open environments, allowing for instance a greater fault tolerance (see [9], [10], [11]).

In this work, we take the perspective of a candidate role player, willing to take part to an interaction, that is ruled by a choreography. We see such an entity as an *enhanced (proactive) service*, made of a declarative description of its behavior, of the operations that it can execute, of its goals [7], and with the capability of reasoning (on its own behavior) so to build *orchestrations* that allow the achievement of its goals (proactiveness). So, for instance, a service that wishes to play the role of the “Seller” of a ticket-purchase choreography may have the aim of “selling tickets”, guaranteed by playing the choreography role, and have the additional, *private goal* of “loyalizing clients”. Notice that personal goals may vary along time, hence, different participations (to the same choreography of a same agent as player of the same role) may be characterized by different personal goals. For example, the flight ticket selling service may change at some point its personal goal, which becomes the “promotion of additional services” offered by the same company. The presence of additional goals, which may not be disclosed to the interacting parties, biases the behavior and the choices of a service. A service will take on a role only after checking the possible achievement of this final condition.

Orchestrations compose the interactions of *sets of services*. However, reasoning on the goals of the orchestrator alone is not sufficient. Indeed, in the context of choreographies that provide a pattern of interaction among services, each service involved has its own goals, that it tries to pursue. The research question is: even though each service has proved that there is a possible way of playing its role, which leads to the achievement of its personal goal, is it possible to guarantee the compatibility of these individual solutions? In other words, will the joint working plan, made of the identified individual solutions, allow the *joint achievement of all the goals* of the interacting parties? A ticket selling service that wants to do some advertisement by sending a newsletter to its clients will

¹REVERSE NoE, <http://reverse.net>

²http://www.w3.org/2005/rules/wiki/RIF_Working_Group

not match the aims of a client which considers this kind of information as spam.

In this paper we describe the first steps of a study about the *joint achievement* of goals of a set of services, whose interaction is ruled by a choreography. The approach relies on a rule-based declarative representation of the choreography roles and of the interacting services, which extends and refines the work in [8], [12], [13], exploiting the results about conservative matching defined in there. The framework enables the application of techniques for reasoning on the effects of playing a role in a choreography, thus checking whether the desired personal goal can be accomplished. The representation is based on the logic programming language described in [14], [15]. Behaviors, as roles and service policies, build upon WSDL-like basic operations, represented as atomic actions with preconditions and effects.

The paper is organized as follows. Section II sets the representation of services and of choreographies that we adopt. Moreover, it explains how it is possible to reason on such a representation in order to allow each service to check the reachability of its goals locally, i.e. by using only the specification of the desired choreography role. Section III tackles the joint achievement of personal goals. A running example is distributed along the pages to better explain the proposed notions and mechanisms. Conclusions end the paper.

II. A THEORETICAL FRAMEWORK FOR REPRESENTING AND REASONING ABOUT SERVICES

In this section, we introduce the notation that we use to represent services and we discuss the problem of verifying a personal goal. The notation, which is a refinement of the proposal in [13], is based on a logical theory for reasoning about actions and change in a *modal logic programming setting* and on the language *DYnamics in LOGic*, described in details in [14]. This language is designed for specifying agents behaviour and for modeling dynamic systems. It is fully set inside the logic programming paradigm by defining programs by sets of Horn-like rules and giving a SLD-style proof procedure. The capability of reasoning about interaction protocols, supported by the language, has already been exploited for customizing web service selection and composition w.r.t. to the user's constraints, based on a semantic description of the services [14]. The language is based on a modal theory of actions and mental attitudes where modalities are used for representing primitive and complex actions as well as the agent beliefs. Complex actions are defined by inclusion axioms [16] and by making use of action operators from dynamic logic, like sequence “;” and test “?”.

In this framework, the problem of reasoning amounts either to build or to traverse a sequence of transitions between *states*. A state is a set of *fluents*, i.e., properties whose truth value can change over time, due to the application of actions. In general, we cannot assume that the value of each fluent in a state is known: we want to have both the possibility of representing unknown fluents and the ability of reasoning about the execution of actions on incomplete states. To explicitly represent

unknown fluents, we use an epistemic operator \mathbf{B} , to represent the beliefs an entity has about the world: $\mathbf{B}f$ means that the fluent f is known to be true, $\mathbf{B}\neg f$ means that the fluent f is known to be false. A fluent f is undefined when both $\neg\mathbf{B}f$ and $\neg\mathbf{B}\neg f$ hold ($\neg\mathbf{B}f \wedge \neg\mathbf{B}\neg f$). Thus each fluent in a state can have one of the three values: *true*, *false* or *unknown*. For the sake of readability, we will add as a superscript of \mathbf{B} the name of the service that has the belief.

Services exhibit interfaces, called port-types, which make a set of operations available to possible clients. In our proposal, a *service description* is defined as a pair $\langle \mathcal{O}, \mathcal{P} \rangle$, where \mathcal{O} is a set of basic operations, and \mathcal{P} (*policy*) is a description of the complex behavior of the service. Analogously to what happens for OWL-S composite processes, \mathcal{P} is built upon basic operations and tests, that control the flow of execution.

A. Basic operations

Let us start with *basic operations*. According to the main languages for representing web services, like WSDL and OWL-S, there are four basic kinds of operations [17] (or atomic processes, when using OWL-S terminology [3]):

- *one-way* involves a single message exchange, a client invokes an operation by sending a message to the service;
- *notify* involves a single message exchange, the client receives a message from the service;
- *request-response* involves the exchange of two messages, are initiated by the invoker of the operation, which sends a message to the service and, after that, waits for a response;
- *solicit-response* involves the exchange of two messages, the order of the messages is inverted w.r.t. a request-response, first the invoker waits for a message from the service and then it sends an answer.

A basic operation is described in terms of its *executability preconditions* and *effects*, the former being a set of fluents (introduced by the keyword **possible if**) which must be contained in the service state in order for the operation to be applicable, the latter being a set of fluents (introduced by the keyword **causes**) which will be added to the service state after the operation execution. Syntax for basic operations is:

$$\text{operation}(\text{content}) \text{ causes } E_s \quad (1)$$

$$\text{operation}(\text{content}) \text{ possible if } P_s \quad (2)$$

where E_s and P_s , denote respectively the fluents, which are expected as effect of the execution of an operation and the precondition to its execution, while *content* denotes possible additional data that is required by the operation. Notice that such operations can also be implemented as invocations to other services.

Operations, when executed, trigger a revision process on the actor's beliefs. Since we describe web services from a *subjective* point of view, we distinguish between the case when the service is either the initiator (the operation *invoker*) or the servant of an operation (the operation *supplier*) by further decorating the operation name with a notation inspired by [18].

The two views are *complementary*, so if one view includes the act of sending a message, the other correspondingly includes a message reception. With reference to a specific service, $operation \gg$ denotes the operation from the point of view of the invoker, while $operation \ll$ denotes the operation from the point of view of the supplier. The view of operations that is used by *invoker* is given in terms of the operation inputs, outputs, preconditions, and effects as usual for semantic web services [3]. In the next part of this section, inputs and outputs are represented as single messages for simplicity but the representation can easily be extended to sets of exchanged data, as in Example (II-C). In this case, preconditions P_s in (2) and effects E_s in (1) are respectively the conditions required by the operation in order to be invoked, and the expected effects that result from the execution of the operation. For what concerns the view of the *supplier*, also in this case the operation is described in terms of its inputs and outputs. Moreover, we also represent a set of conditions that enable the executability of the operation. In order to distinguish them from the above, in this case we use R_s instead of P_s in (2), calling them *requirements*. Finally, we represent a set of conditions that constitute the *side effects* of the operation. In this case we use S_s instead of E_s in (1). For example, a *buy* operation of a selling service has as a precondition the fact that the invoker has a valid credit card, as inputs the credit card number of the buyer and its expiration date, as output it generates a receipt, and as effect the credit card is charged. From the point of view of the supplier, the requirement to the execution is to have an active connection to the bank, and the side effect is that the store availability is decreased while the service bank account is increased of the perceived amount.

Let us now introduce the formal representation of the four kinds of basic operations (for each operation we report both views) and of complex operations.

1) *One-way*: In one-way operations, the *invoker* requests an execution which involves sending an information m_{in} to the supplier; the invoker must obviously know the information to send before the invocation (a) (see Table I). The invoker can execute the operation only if the preconditions are satisfied in its current state (a). The execution of the invocation brings about the effects E_s (c), and the invoker will know that it has sent an information to the supplier (b). Using OWL-S terminology, m_{in} represents the *input* of the operation, while P_s and E_s are its preconditions and effects. One-way operations have no output. On the other hand, the *supplier*, which exhibits the one-way operation as one of the services that it can execute, has the requirements R_s (e). The execution of the operation causes the supplier to know the information sent by the invoker (f). We also allow the possibility of having some side effects on the supplier's state. These effects are not to be confused with the operation effects described by IOPE, and have been added for the sake of completeness.

2) *Notify*: With ref. to Table I, in notify operations, the *invoker* requests an execution which involves receiving an information m_{out} from the supplier. The invoker can execute the operation only if the preconditions are satisfied in its

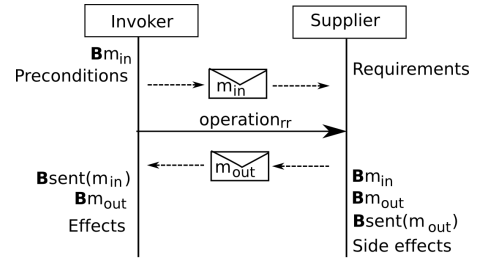


Fig. 1. The request-response basic operation.

current state (a). The execution of the invocation brings about the effects E_s (c), and the invoker will know the received information (b). Using OWL-S terminology, m_{out} represents the *output* of the operation, while P_s and E_s are its preconditions and effects. Notify operations have no input. The *supplier* must meet the requirements R_s (e). The execution of the operation simply causes the fact that the supplier will know the message to send (f) and that it has sent some information to the invoker (g). As above, we allow the possibility of having some side effects on the supplier's state (h).

3) *Request-response*: In request-response operations (see Figure 1), the *invoker* requests an execution which involves sending an information m_{in} (the input, according to OWL-S terminology) and then receiving an answer m_{out} from the supplier (the output in OWL-S). The invoker can execute the operation only if the preconditions P_s are satisfied in its current state and if it owns the information to send (a) (see Table I). The execution of the invocation brings about the effects E_s (d), and the fact that the invoker knows that it has sent the input m_{in} to the supplier (b). One further effect of the execution is that the invoker knows the answer returned by the operation (c). This representation abstracts away from the actual message exchange mechanism, which is implemented. Our aim is to reason on the effects of the execution on the mental state of the parties [15]. As for one-way operations, the *supplier* has the requirements R_s to the operation execution (e). It receives an input m_{in} from the invoker (f). The execution produces an answer m_{out} (g), which is sent to the invoker (h). As usual, it is possible to have some side effects on the supplier's state. On the supplier's side, we can notice more evidently the abstraction of the representation from the actual execution process. In fact, we do not model how the answer is produced but only the fact that it is produced.

4) *Solicit-response*: With ref. to Table I, in solicit-response operations, the *invoker* requests an execution which involves receiving an information m_{out} (the output, according to OWL-S terminology) and then sending a message m_{in} to the supplier (the input in OWL-S). The invoker can execute the invocation only if the preconditions P_s are satisfied in its current state (a). The execution of the invocation brings about the effects E_s (e). The invoker receives a message m_{out} from the supplier (b) then, it produces the input information m_{in} which is sent to the supplier, see (c) and (d). As for notify operations, the

Operation	Invoker's view	Supplier's view
One-Way	(a) $\text{operation}_{ow}^{\gg}(m_{in})$ possible if $B^{Invoker} m_{in} \wedge P_s$ (b) $\text{operation}_{ow}^{\gg}(m_{in})$ causes $B^{Invoker} sent(m_{in})$ (c) $\text{operation}_{ow}^{\gg}(m_{in})$ causes E_s	(e) $\text{operation}_{ow}^{\ll}(m_{in})$ possible if R_s (f) $\text{operation}_{ow}^{\ll}(m_{in})$ causes $B^{Supplier} m_{in}$ (g) $\text{operation}_{ow}^{\ll}(m_{in})$ causes S_s
Notify	(a) $\text{operation}_n^{\gg}(m_{out})$ possible if P_s (b) $\text{operation}_n^{\gg}(m_{out})$ causes $B^{Invoker} m_{out}$ (c) $\text{operation}_n^{\gg}(m_{out})$ causes E_s	(e) $\text{operation}_n^{\ll}(m_{out})$ possible if R_s (f) $\text{operation}_n^{\ll}(m_{out})$ causes $B^{Supplier} m_{out}$ (g) $\text{operation}_n^{\ll}(m_{out})$ causes $B^{Supplier} sent(m_{out})$ (h) $\text{operation}_n^{\ll}(m_{out})$ causes S_s
Request-response	(a) $\text{operation}_{rr}^{\gg}(m_{in}, m_{out})$ possible if $B^{Invoker} m_{in} \wedge P_s$ (b) $\text{operation}_{rr}^{\gg}(m_{in}, m_{out})$ causes $B^{Invoker} sent(m_{in})$ (c) $\text{operation}_{rr}^{\gg}(m_{in}, m_{out})$ causes $B^{Invoker} m_{out}$ (d) $\text{operation}_{rr}^{\gg}(m_{in}, m_{out})$ causes E_s	(e) $\text{operation}_{rr}^{\ll}(m_{in}, m_{out})$ possible if R_s (f) $\text{operation}_{rr}^{\ll}(m_{in}, m_{out})$ causes $B^{Supplier} m_{in}$ (g) $\text{operation}_{rr}^{\ll}(m_{in}, m_{out})$ causes $B^{Supplier} m_{out}$ (h) $\text{operation}_{rr}^{\ll}(m_{in}, m_{out})$ causes $B^{Supplier} sent(m_{out})$ (i) $\text{operation}_{rr}^{\ll}(m_{in}, m_{out})$ causes S_s
Solicit-response	(a) $\text{operation}_{sr}^{\gg}(m_{in}, m_{out})$ possible if P_s (b) $\text{operation}_{sr}^{\gg}(m_{in}, m_{out})$ causes $B^{Invoker} m_{out}$ (c) $\text{operation}_{sr}^{\gg}(m_{in}, m_{out})$ causes $B^{Invoker} m_{in}$ (d) $\text{operation}_{sr}^{\gg}(m_{in}, m_{out})$ causes $B^{Invoker} sent(m_{in})$ (e) $\text{operation}_{sr}^{\gg}(m_{in}, m_{out})$ causes E_s	(f) $\text{operation}_{sr}^{\ll}(m_{in}, m_{out})$ possible if R_s (g) $\text{operation}_{sr}^{\ll}(m_{in}, m_{out})$ causes $B^{Supplier} m_{out}$ (h) $\text{operation}_{sr}^{\ll}(m_{in}, m_{out})$ causes $B^{Supplier} sent(m_{out})$ (i) $\text{operation}_{sr}^{\ll}(m_{in}, m_{out})$ causes $B^{Supplier} m_{in}$ (l) $\text{operation}_{sr}^{\ll}(m_{in}, m_{out})$ causes S_s

TABLE I

THE REPRESENTATION OF THE FOUR KINDS OF BASIC OPERATIONS IN THE PROPOSED FRAMEWORK: EACH OPERATION IS GIVEN IN TERMS OF ITS PRECONDITIONS AND EFFECTS, FOR EACH OF THEM WE REPORT BOTH THE INVOKER'S VIEW AND THE SUPPLIER'S VIEW.

supplier must fulfill the requirements R_s (f). The execution causes the supplier to know the information to send (g) and that it has sent such information to the invoker (h). Moreover, it produces also the knowledge of the information m_{in} received from the invoker (i). Side effects on the supplier's state are allowed (l).

B. Service policies and choreography roles

The framework accounts also for *complex behaviors* that require the execution of many operations. A service policy \mathcal{P} is a collection of clauses of the kind:

$$p_0 \text{ is } p_1, \dots, p_n \quad (3)$$

where p_0 is the name of the procedure and p_i , $i = 1, \dots, n$, is either an atomic action (operation), a test action (denoted by the symbol $?$), or a procedure call. Procedures can be recursive and are executed in a goal-directed way, similarly to standard logic programs, and their definitions can be non-deterministic as in Prolog. Complex behaviors are used also for representing choreography roles. Generally speaking, we represent a *choreography* \mathcal{C} as a tuple (R_1, \dots, R_n) of interacting and complementary *roles*, each role R_i being a subjective view of the interaction that is encoded.

Also a role R is represented by a pair $\langle \mathcal{O}, \mathcal{P} \rangle$. The difference with services is that it composes *specifications* of operations and not implemented operations. The player of each role has to supply appropriate implementations. We call such specifications *unbound operations*. Formally, unbound

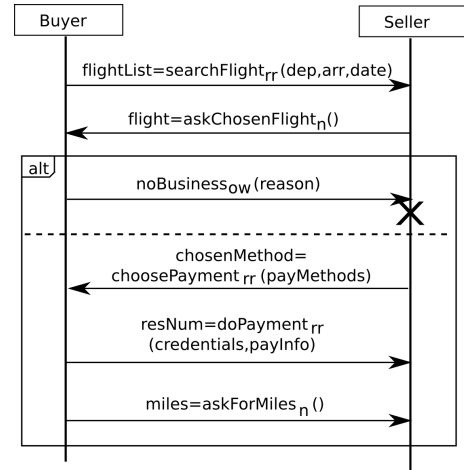


Fig. 2. A simple choreography for reserving a flight, expressed as a UML sequence diagram.

operations are represented as normal basic operations, in terms of their preconditions and effects. This is one of the advantages of adopting a logic language: it allows handling implementations and specifications in the same way. In our case, both operations and operation specifications are given in terms of their preconditions and effects.

C. Flight-purchase

As an example, let's consider `searchFlight`, an operation of a flight reservation service, which is offered by a *seller* and can be invoked by a *buyer* to search information about flights with given departure (*dep*) and arrival locations (*arr*) plus the date of departure (*date*). From the point of view of the buyer, the operation, which is of kind request-response, is:

- (a) $\text{searchFlight}_{rr}^{\gg}((dep, arr, date), flightList)$ **possible if**
 $\mathbf{B}^{buyer} dep \wedge \mathbf{B}^{buyer} arr \wedge \mathbf{B}^{buyer} date \wedge$
 $\mathbf{B}^{buyer} \neg \text{sellingStarted}$
- (b) $\text{searchFlight}_{rr}^{\gg}((dep, arr, date), flightList)$ **causes**
 $\mathbf{B}^{buyer} sent(dep) \wedge \mathbf{B}^{buyer} sent(arr) \wedge$
 $\mathbf{B}^{buyer} sent(date)$
- (c) $\text{searchFlight}_{rr}^{\gg}((dep, arr, date), flightList)$ **causes**
 $\mathbf{B}^{buyer} flightList$
- (d) $\text{searchFlight}_{rr}^{\gg}((dep, arr, date), flightList)$ **causes**
 $\mathbf{B}^{buyer} \text{sellingStarted}$

The inputs of the operation are *dep*, *arr*, and *date*, while the output is *flightList*. In this case the set P_s contains only the belief $\mathbf{B}^{buyer} \neg \text{sellingStarted}$ (in bold text above) while the set E_s of effects contains the belief $\mathbf{B}^{buyer} \text{sellingStarted}$ (in bold text as well).

From the point of view of the supplier, instead, the operation is represented as:

- (a) $\text{searchFlight}_{rr}^{\ll}((dep, arr, date), flightList)$ **possible if**
true
- (b) $\text{searchFlight}_{rr}^{\ll}((dep, arr, date), flightList)$ **causes**
 $\mathbf{B}^{seller} dep \wedge \mathbf{B}^{seller} arr \wedge \mathbf{B}^{seller} date$
- (c) $\text{searchFlight}_{rr}^{\ll}((dep, arr, date), flightList)$ **causes**
 $\mathbf{B}^{seller} flightList$
- (d) $\text{searchFlight}_{rr}^{\ll}((dep, arr, date), flightList)$ **causes**
 $\mathbf{B}^{seller} sent(flightList)$

In this case the sets R_s and S_s of requirements and side effects are empty. The operation expects as input the departure and arrival locations and the date of the flight, and it produces a *flightList*, which it sends to its customer, so after the operation the belief $\mathbf{B}^{seller} sent(flightList)$ will be in its belief state.

`buyTicket` is, instead, an example of procedure implemented by a possible buyer service:

- (a) `buyTicket` **is**
 $\text{searchFlight}_{rr}^{\gg}((dep, arr, date), flightList);$
 $\text{askChosenFlight}_n^{\ll}(flight);$
`evaluateAndBuy.`
- (b) `evaluateAndBuy` **is**
 $\text{noBusiness}_{ow}^{\gg}(reason).$
- (c) `evaluateAndBuy` **is**
 $\text{choosePayment}_{rr}^{\ll}(payMethods, chosenMethod);$
 $\text{doPayment}_{rr}^{\gg}((credential, payInfo), resNum);$
 $\text{askForMiles}_n^{\gg}(miles).$

First, it invokes an operation for searching flights that correspond to a given specification ($\text{searchFlight}_{rr}^{\gg}$). After that,

it waits for being asked about the preferred flight, chosen among the flights list obtained by the previous operation ($\text{askChosenFlight}_n^{\ll}$). This evaluation can give either a negative outcome, hence the interaction is interrupted ($\text{noBusiness}_{ow}^{\gg}$) or the interaction continues with the selection of the payment method ($\text{choosePayment}_{rr}^{\ll}$). Then, the buyer performs the payment ($\text{doPayment}_{rr}^{\gg}$) and, at the end, it is notified about the obtained miles ($\text{askForMiles}_n^{\gg}$).

D. Reasoning on goals

In the outlined framework, it is possible to reason about personal goals by means of queries of the form *Fs after p*, where *Fs* is the *goal* (represented as a conjunction of fluents), that we wish to hold after the execution of a policy *p*. Checking if a formula of this kind holds corresponds to answering the query: “Is it possible to execute *p* in such a way that the condition *Fs* is true in the final state?”. When the answer is positive, the reasoning process returns a sequence of atomic actions that allows the achievement of the desired condition. This sequence corresponds to an execution trace of the procedure and can be seen as a *plan* to bring about the goal *Fs*. This form of reasoning is known as *temporal projection*. Temporal projection fits our needs because, as mentioned in the introduction, in order to perform the selection we need a mechanism that verifies if a goal condition holds after the interaction with the service has taken place. *Fs* is the set of facts that we would like to hold “after” *p*.

Reasoning is done by exploiting a goal-directed proof procedure (denoted by “ \vdash ” in the following) designed for the language *Dynamics in LOGIC* [15], [14], which supports both temporal projection and planning and allows the proof of existential queries of the kind reported above. The procedure definition constrains the search space. In particular, for what concerns planning, the proof procedure allows the automatic extraction of linear or conditional plans for achieving the goal of interest from an incompletely specified initial state.

Let $\langle \mathcal{O}, \mathcal{P} \rangle$ be a service description. The application of temporal projection to \mathcal{P} returns, if any, an execution trace (a linear plan), that makes a goal of interest become true. Let us, then, consider a procedure *p* belonging to \mathcal{P} and denoting its top level, and denote by *Q* the query *Fs after p*. Given a state s_0 , containing all the fluents that we know as being true in the beginning, we denote the fact that the query *Q* is successful in the service description by $(\langle \mathcal{O}, \mathcal{P} \rangle, s_0) \vdash Q$. The execution of the above query returns as a side-effect an *execution trace* σ of *p*; the execution trace σ is a sequence a_1, \dots, a_n of atomic actions. We denote this by:

$$(\langle \mathcal{O}, \mathcal{P} \rangle, s_0) \vdash Q \text{ w.a. } \sigma \quad (4)$$

where “w.a.” stands for *with answer*.

For example, suppose that the initial state of the service *b1* is $s_0 = \{\mathbf{B}^{buyer} dep, \mathbf{B}^{buyer} arr, \mathbf{B}^{buyer} date, \mathbf{B}^{buyer} deferredPaymentPos, \mathbf{B}^{buyer} \neg \text{sellingStarted}, \mathbf{B}^{buyer} credentials\}$, (all the other fluents truth value is “unknown”). This means that *b1* assumes a date, a departure location, an arrival location, the fact that it is

possible to defer the payment to the departure (at a desk at the airport), and that no selling process has started yet. The goal of $b1$ is to achieve the following condition: $Q = \{\mathbf{B}^{buyer}sellingComplete, \mathbf{B}^{buyer}resNum\}$ after buyTicket. Intuitively, the buyer expects that, after the interaction, it will have a reservation number as a result.

By reasoning on its policy and by using the definitions of the unbound operations that are given by the choreography, $b1$ can identify an execution trace, that leads to a state where Q holds:

$$\begin{aligned} \sigma = & \text{searchFlight}_{rr}^{\gg}((dep, arr, date), flightList); \\ & \text{askChosenFlight}_n^{\ll}(flight); \\ & \text{choosePayment}_{rr}^{\ll}(payMethods, chosenMethod); \\ & \text{doPayment}_{rr}^{\gg}((credential, payInfo), resNum); \\ & \text{askForMiles}_n^{\gg}(miles) \end{aligned}$$

This is possible because in a declarative representation specifications are executable. Moreover notice that this execution does not influence the belief about the deferred payment, which persists from the initial through the final state and is not contradicted.

III. JOINT ACHIEVEMENT OF PERSONAL GOALS

So far, we have talked about goals, whose achievement motivates the participation of a service to a choreography. It is, in fact, reasonable that a service may decide of taking on a role only if by doing so it will have the possibility of satisfying its purposes. In our proposal a service takes this decision based on knowledge that includes a public role representation and a representation, given in terms of preconditions and effects, of its own operations, part of which will substitute some specifications contained in the role description. More generally, a choreography is made of many roles, that are played by different services, each of which has its own goal. The question we try to answer here is whether it is possible for this team of enhanced proactive services to reach an agreement about their possible executions so that in the team all of them will achieve their personal goals. As observed in [19], [20] the team can achieve a goal if there is a *joint working plan* that can achieve the goal.

For example, consider a choreography $\mathcal{C} = (R_1, R_2)$ and two services, interested to play respectively R_1 and R_2 for achieving the goals G_1 and G_2 . By applying the described reasoning technique, the two services might both find that they can achieve their personal goals, one by following the execution trace σ_1 , the other σ_2 . The problem is that σ_1 and σ_2 might not be compatible, e.g. one invokes a *operation* $^{\gg}$, when the other side does not include the corresponding execution of *operation* $^{\ll}$.

In some cases, such compatible execution traces might not exist. In others, each party may have a set of alternative σ_i available, part of which are indeed compatible with executions traces identified by the partner. The problem, then, becomes to converge to a common solution that allows for the achievement of both goals. Let us discuss these issues, focusing on two-

role choreographies. To this aim, let us introduce a few useful notions.

Definition 1 (Compatible execution traces): Let $\sigma = a_0; \dots; a_n$ and $\sigma' = a'_0; \dots; a'_n$ be two execution traces, we say that σ is *compatible* to σ' iff for each operation a_i in σ the corresponding a'_i in σ' is its complementary view.

Given an operation a , we denote by \bar{a} its complementary operation. For instance, in Example II-C $\text{searchFlight}_{rr}^{\gg}$ is complementary to $\text{searchFlight}_{rr}^{\ll}$ and vice versa. We extend the notion of complementarity to execution traces, so the complementary $\bar{\sigma}$ on an execution trace σ is the sequence made of the operations that are respectively complementary to those of σ .

When a service plays a choreography role, it must supply a set of operations that will substitute the corresponding specifications, thus producing a service policy. Let $\langle \mathcal{O}, \mathcal{P} \rangle$ be a service description, and let \mathcal{O}_u^i be a subset of \mathcal{O} , containing unbound operations that are to be supplied by a same role player S_i . Let \mathcal{O}_{S_i} be the set of such operations, that are bound to the operations in \mathcal{O}_u^i . In case S_i is the player of $\langle \mathcal{O}, \mathcal{P} \rangle$, they will be operations decorated by \ll , otherwise they will be \gg operations. We represent the binding by the substitution $\theta = [\mathcal{O}_{S_i}/\mathcal{O}_u^i]$ applied to $\langle \mathcal{O}, \mathcal{P} \rangle$. Notice that by $[\mathcal{O}_{S_i}/\mathcal{O}_u^i]$ we identify a set of substitutions $[o/o_u]$ of single service operations to single unbound operations. The application of the substitution is denoted by $\langle \mathcal{O}\theta, \mathcal{P}\theta \rangle$, where every element of \mathcal{O}_u^i is substituted by/bound to an element of \mathcal{O}_{S_i} .

When the matching process is applied for selecting a service that should play a role in a choreography, the desire is that *the substitution* (of the service operations to the specifications contained in the choreography) *preserves the properties of interest*, i.e. the goals that could be entailed before by reasoning on each role description separately should be still achievable. Thus, we rely on the notion of *Conservative substitution* by Baldoni et al. in [13].

Let us, now, consider two-role choreography and see how the services S_1 and S_2 can identify a set of compatible traces, whose execution allows the achievement of both their personal goals. The mechanism we are going to describe can be extended to more complicated choreographies. However, we do not discuss the case, due to the lack of space, concerning a number of roles more than two. Suppose that S_1 has identified an execution trace σ that allows the achievement of its goal F_{S_1} , and it has verified that the substitutions θ^{R_1} and θ^{R_2} are conservative (the two substitutions involve disjoint sets of unbound operations by construction). Therefore, it now has an execution trace $\sigma\theta^{R_1}\theta^{R_2}$ that does not contain unbound operations. Before executing it, its candidate partner S_2 must agree on executing the complementary trace $\bar{\sigma}\theta^{R_1}\theta^{R_2}$. Of course, S_2 might have its own goal F_{S_2} which should be achieved with the execution of the top level procedure of the role R_2 , that we here call p_2 . Therefore, the following conditions must be verified:

- 1) $\bar{\sigma}\theta^{R_1}\theta^{R_2}$ must be an execution trace of p_2 ;
- 2) after executing $\bar{\sigma}\theta^{R_1}\theta^{R_2}$ the goal F_{S_2} must hold.

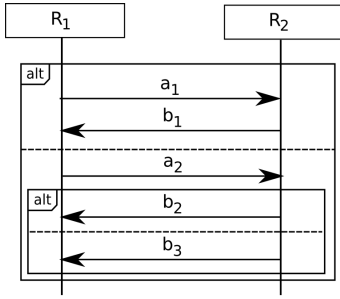


Fig. 3. Role R_1 takes a decision about invoking operation a_1 or a_2 on R_2 . In the former case, R_2 will subsequently invoke b_1 over R_1 ; in the latter, it will take a decision and choose between the invocation of b_2 or the invocation of b_3 on R_1 .

The first condition is guaranteed by the assumption that the choreography is well-defined, i.e. roles are interoperable, and that the services that will animate it are conform to the corresponding roles. The expectation that the roles of a choreography are by construction interoperable and that services are conform to them implies that, for any execution trace that a party can execute, its partner has a complementary execution trace. Interoperability is a hot research issue. A discussion about it is out of the scope of this work but the interested reader can take a look at [21], [22], [23], [24].

The second condition is to be verified by S_2 by reasoning on its goal, i.e. by verifying that $\langle R_2 \theta^{R_1} \theta^{R_2}, s_0 \rangle \vdash F_{S_2}$ after $\bar{\sigma} \theta^{R_1} \theta^{R_2}$, where s_0 in this case is the initial state of S_2 and $\theta^{R_1} \theta^{R_2}$ represents the substitution obtained from $\theta^{R_1} \theta^{R_2}$ by using the complementary views of the involved operations. Notice that the above reasoning is much simpler than the one executed on p_1 because S_2 only has to check if the goal is satisfied after $\bar{\sigma}$ has been executed. The reasoning applied to p_1 , instead, was aimed at identifying such a trace.

To increase the probability of reaching an agreement, we can imagine that S_1 produces a set of alternative execution traces Σ , each of which allows the achievement of F_{S_1} , and then propose them to S_2 , that will restrict them to a set of alternatives $\Sigma' \subseteq \Sigma$, satisfying both goals. It is worth noting that, in general, the set of alternative execution traces might not be finite. Moreover, the derivation (\vdash) is semi-decidable unless the choreographies are properly restricted, for instance by focussing onto regular sets [15]. Indeed, many choreographies are of this kind [25]. This interaction between the services, proposed so to allow the joint achievement of their goals, can be seen as a kind of *negotiation* [26], [27]. In case no other preference criterion is introduced, it is *indifferent* which of the execution traces in Σ' will be followed, and it is not necessary to perform any further negotiation step because the choreography is respected by both partners [24], and Σ' is known to both of them. Whatever the initiator of the interaction will be, they know how to behave to allow the mutual achievement of their personal goals, although none of them can make a commitment on a specific execution trace because both take choices that contribute to its definition.

An interesting issue is whether there is any *guarantee* that both partners will stick to execution traces that are in Σ' . In game theory words [26] and considering a choreography as a set of game rules, is Σ' an *equilibrium*? Generally, the answer is no. It might, in fact, be the case that S_2 has some execution trace, that is not contained in Σ' but allows anyway the achievement of its goal: at some point of the execution of an agreed trace, S_2 might decide to continue with an unagreed action, because such action is particularly convenient for it. A related issue is whether a service has a *dominant strategy*, i.e. an execution trace such that all of the alternative courses of action, that its partner has, belong to Σ' . For instance, with reference to Figure 3, let us suppose that Σ' contains the execution traces $a_1; b_1$ and $a_2; b_2$. In this case, the execution trace $a_1; b_1$ is dominant for S_1 because the only possible answer of S_2 is b_1 and the overall trace allows the achievement of both goals. Instead, the execution trace $a_2; b_2$ is not dominant because S_2 has the possibility of deciding to execute b_3 in alternative to b_2 and the trace $a_2; b_3$ does not belong to Σ' . The existence of a dominant strategy depends on the goal that one wants to satisfy, on the choreography, and on the identified substitutions. The derivation that we have proposed is not focussed on the identification of dominant strategies. In general, and differently than what happens for equilibria, the decision whether an execution trace is a dominant strategy can be taken without knowing the goal of the interlocutor. Indeed, a strategy is dominant if any action that the interlocutor can perform, lead to an execution trace which still belong to Σ' . Thus a service which has a dominant strategy has the guarantee to have a way to reach his goal. Therefore, it can be taken by a service individually.

Another interesting problem concerns preference criteria that services may apply in order to rank a set of strategies. Supposing that the two services decide to behave well and to execute a trace which is in the agreement, how can they converge to a best-compromise agreement? One criterion could be to select the trace which entails the minimum number of effects. For example, one can imagine that a buyer of a flight ticket prefers a trace at the end of which it has simply bought the desired ticket w.r.t. one in which it has additionally been registered in the advertisement mailing list of the flight company. This issue will be part of our future works.

As a final remark, negotiation, *does not substitute* the execution of the choreography but it rather concerns checking the possibility of, in this case, achieving all of the goals. The fact that the two services converged to a set of promising execution traces does not imply that, after starting the real execution, they will be able to stick to them. The reason is that those traces were identified by making assumptions on the values returned by tests and conditions, which cannot be known in advance. The actual results of such tests, obtained at execution time, could be different than the assumed ones. The *additional value* of performing the negotiation is double: on the one hand, it is possible to avoid the interaction with partners with which the agreement cannot be reached at all, on the other hand, each partner has the means to understand when

the interaction takes an unagreed path and decide accordingly, for instance concluding that it is better to stop the interaction.

IV. CONCLUSIONS

The coordination of a set of autonomous, cooperating agents is well-known and crucial in multi-agent systems research [27]. Solutions proposed in the literature are, for example, coordination as a *post-planning process* [28], where constraints are checked after the plan has been found, and the use of *conversation moderators* [29], which guarantee that achievement of shared objectives.

Along this line, this work tackles the problem of allowing a set of independent services, which must cooperate in the context of a given choreography, to reason about the joint achievement of their goals. The approach that we propose implements a simple form of *negotiation*, inspired by [20], [19]. There are, however, some important differences w.r.t the work by Ghaderi et al. The first is that the behavior of our services is ruled by a choreography, which specifies all the possible interactions that can occur. The assumption is that when services play choreography roles, they commit to keep their behavior adherent to the role specification. Ghaderi et al., instead, do not use or represent choreographies or other kinds of interaction protocols: the reasoning process considers all the possible execution traces that can be composed out of a set of atomic actions. Moreover, even when an execution trace, that allows the joint achievement of goals, is identified there is still the need of adding coordination mechanisms that allow its actual execution. In our case, the necessary coordination that makes the cooperation of the services possible is supplied by the choreography. The other assumption that we make is that the roles that compose a choreography are interoperable. Interoperability is a hot research topic that is orthogonal to the problems faced in this work.

Moreover, in the work by Ghaderi et al. each agent reasons also for the partner, because the two agents share a common goal and a common state. The method, when successful, identifies a set of joint plans, that correspond to preferred strategies for the agents. These are obtained by the iterative elimination of dominated strategies. In our framework each partner does not have knowledge about the goals of its interlocutor, as it is reasonable to suppose for web services; therefore, the execution traces that we identify are not necessarily dominant. For example, a greedy partner may take an action (if the protocol includes it) that is not in the agreement but that allows the immediate achievement of its own goal. This behavior is, however, not convenient over the long run because the former partner knows the choreography and knows the agreed traces, therefore, it has a way of *monitoring* the on-going course of interaction. As soon as it realizes that the partners has deviated from the agreed traces, it can take appropriate action, e.g. interrupt the interaction, enact compensation actions, publish a low reputation rate for the partner. In other words, there is a correspondence to what is known in game theory as a game iterated forever [27].

Some correspondences with the technique “Planning as Model Checking” proposed in [30] can be investigate. In this approach, the planning domain is a semantic model, the planning problem is represented through a set of global state and plan generation is done by checking whether suitable formulas are true in a semantic model. However, deal with web services that have to cooperate in order to reach their own goals seems to be more complicated. Indeed, planning has to be performed by each service on its own, ignoring which are the goals and which will be the substitutions used by the interlocutors. At the end, the chosen plan must be convenient for every service involved. Instead, by means of the mechanism proposed in this paper planning is made only by one service and the resulting set of compatible plans, at the end of the negotiation, allows everyone to achieve its own goal.

Some similarities can be found also with the proposal by Son and Sakama [31], that concerns the process of creating a *joint plan* in a MAS. They define a joint plan as one in which there are some *cooperative actions*, i.e. actions that are mutually requested/offered by agents. Indeed, also the actions involved in a protocol can be seen as cooperative actions, in the sense that a service is not able to achieve its goal on its own, and needs the “cooperation” of other services. The main difference between the two proposals is that a protocol defines not only which actions can be invoked on other services, but also when this can occur. One of the advantages of choreographies is that they limit the number of possible plans, restricting the search space of a joint plan. Moreover, condition 1) in Section III is guaranteed by the fact that whatever execution trace one of the partners focuses on, the other surely has a compatible one, if all of the interacting partners are conformant to the choreography and this is made of a set of interoperable roles.

Some analogies can be found also with the case in which protocols are defined by *social commitments* [32]. Indeed, a commitment is a “promise” of an agent to another on performing some actions. A protocols defined by commitments does not specify when actions are to be taken. However, in this case there is a common knowledge (i.e. the commitments) that constrain somehow the behaviour of the involved parties. This mechanism can guide the reasoning process in a better way. Indeed the reasoning is not on the belief that one agent has on the others –e.g. I believe that if he has provide this actions he will do it–, but is on the obligation one agent has taken: I know that he has taken the obligation of performing this action when I will ask him. Of course, this does not mean that the action will be executed when required. Indeed, actions are defined in terms of preconditions. Thus, its is possible that a condition is not satisfied when the action should be executed.

REFERENCES

- [1] W3C, “Web services description language (WSDL) version 2.0 part 1: Core language.” [Online]. Available: <http://www.w3.org/TR/wsdl20/>
- [2] OASIS, “Web services business process execution language version 2.0, working draft.” [Online]. Available: <http://www.oasis->

- open.org/committees/download.php/17355/wsbpel-specification-draft.doc
- [3] OWL-S Coalition. [Online]. Available: <http://www.daml.org/services/owl-s/>
- [4] D. Fensel, H. Lausen, J. de Bruijn, M. Stollberg, D. Roman, and A. Polleres, *Enabling Semantic Web Services : The Web Service Modeling Ontology*. Springer.
- [5] B. Meyer, "Applying "design by contract"," *Computer*, vol. 25, no. 10, pp. 40–51, 1992.
- [6] M. Pistore, L. Spalazzi, and P. Traverso, "A minimalist approach to semantic annotations for web processes compositions." in *ESWC*, ser. LNCS, Y. Sure and J. Domingue, Eds., vol. 4011. Springer, 2006, pp. 620–634.
- [7] M. B. van Riemsdijk and M. Wirsing, "Goal-Oriented and Procedural Service Orchestration. A Formal Comparison," in *AWESOME07*, Durham, UK, September 2007.
- [8] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella, "Reasoning on choreographies and capability requirements," *International Journal of Business Process Integration and Management*, vol. 2, no. 4, pp. 247–261, 2007.
- [9] G. Caire, D. Gotta, and M. Banzi, "WADE: A software platform to develop mission critical applications exploiting agents and workflows," in *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Estoril, Portugal, 12-16 May 2008, pp. 29–36.
- [10] M. Piunti, A. Ricci, and A. Santi, "SOA/WS Applications using Cognitive Agents working in CArTAgO Environments," in *Decimo Workshop Nazionale "Dagli Oggetti agli Agenti" (WOA 2009)*, Parma, 2009.
- [11] L. Bozzo, V. Mascardi, D. Ancona, and P. Busetta, "CooWS: Adaptive BDI agents meet service-oriented computing," in *Proceedings of the Int. Conference on WWW/Internet*, 2005, pp. 205–209.
- [12] M. Baldoni, C. Baroglio, A. Martelli, V. Patti, and C. Schifanella, "Service selection by choreography-driven matching," in *Emerging Web Services Technology*, ser. Whitestein Series in Software Agent Technologies and Autonomic Computing, T. Gschwind and C. Pautasso, Eds. Birkhäuser, September 2008, vol. II, ch. 1, pp. 5–22.
- [13] M. Baldoni, C. Baroglio, V. Patti, and C. Schifanella, "Conservative reuse ensuring matches for service selection," in *Proc. of the 6th European Workshop on Multi-Agent Systems (EUMAS)*, 2008.
- [14] M. Baldoni, C. Baroglio, A. Martelli, and V. Patti, "Reasoning about interaction protocols for customizing web service selection and composition," *JLAP, special issue on Web Services and Formal Methods*, vol. 70, no. 1, pp. 53–73, 2007.
- [15] M. Baldoni, L. Giordano, A. Martelli, and V. Patti, "Programming Rational Agents in a Modal Action Logic," *Annals of Mathematics and Artificial Intelligence, Special issue on Logic-Based Agent Implementation*, vol. 41, no. 2-4, pp. 207–257, 2004. [Online]. Available: <http://www.kluweronline.com/issn/1012-2443>
- [16] M. Baldoni, "Normal Multimodal Logics: Automatic Deduction and Logic Programming Extension," Ph.D. dissertation, Dipartimento di Informatica, Università degli Studi di Torino, Italy, 1998.
- [17] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services*. Springer, 2004.
- [18] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella, "Synthesis of Underspecified Composite e-Service bases on Automated Reasoning," in *Proc. of ICSOC04*. ACM, 2004, pp. 105–114.
- [19] H. Ghaderi, H. Levesque, and Y. Lespérance, "Towards a logical theory of coordination and joint ability," in *Proc. of the 6th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS07)*, 2007, pp. 532–534.
- [20] —, "A logical theory of coordination and joint ability," in *Proc. of AAAI'07*, 2007, pp. 421–426.
- [21] S. K. Rajamani and J. Rehof, "Conformance checking for models of asynchronous message passing software," in *CAV*, ser. LNCS, vol. 2404. Springer, 2002, pp. 166–179.
- [22] L. Bordeaux, G. Salaün, D. Berardi, and M. Mecella, "When are two web services compatible?" in *TES 2004*, ser. LNCS, vol. 3324. Springer, 2005, pp. 15–28.
- [23] M. Bravetti and G. Zavattaro, "Contract based multi-party service composition," in *FSEN*, ser. LNCS, vol. 4767. Springer, 2007, pp. 207–222.
- [24] M. Baldoni, C. Baroglio, A. Chopra, N. Desai, V. Patti, and M. Singh, "Choice, interoperability, and conformance in interaction protocols and service choreographies," in *Proc. of the 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, 2009.
- [25] Foundation for Intelligent Physical Agents, "<http://www.fipa.org>."
- [26] M. J. Osborne and A. Rubinstein, *A Course in Game Theory*. MIT Press, July 1994.
- [27] M. Wooldridge, *An Introduction to Multiagent Systems*. John Wiley & Sons, March 2002.
- [28] J. R. Steenhuisen, C. Witteveen, A. ter Mors, and J. Valk, "Framework and Complexity Results for Coordinating Non-cooperative Planning Agents," in *MATES*, ser. Lecture Notes in Computer Science, vol. 4196. Springer, 2006, pp. 98–109.
- [29] C. Sibertin-Blanc and N. Hameurlain, "Participation Components for Holding Roles in Multiagent Systems Protocols," in *Engineering Societies in the Agents World*, ser. Lecture Notes in Computer Science, vol. 3451, August 2005, pp. 60–73.
- [30] F. Giunchiglia and P. Traverso, "Planning as Model Checking," in *Recent Advances in AI Planning*, ser. LNCS. Springer, 2000, pp. 1–20.
- [31] T. C. Son and C. Sakama, "Reasoning and Planning with Cooperative Actions for Multiagents Using Answer Set Programming," in *DALT: Declarative Agent Languages and Technologies*, ser. LNAI. Budapest, Hungary: Springer, May 2009.
- [32] M. P. Singh, "Agent communication languages: Rethinking the principles," in *Communication in Multiagent Systems*, ser. Lecture Notes in Computer Science, M.-P. Huget, Ed., vol. 2650. Springer, 2003, pp. 37–50.

Verifying A-Priori the Composition of Declarative Specified Services.

Federico Chesani, Paola Mello, Marco Montali, Paolo Torroni

Abstract—Service Oriented Architectures are knowing a wide success, thanks to the maturity of standards and implementations. Moreover, the possibility of composing complex systems starting from simpler services is becoming supported by industrial tools, although still immature at the standard level. However, the a-priori verification aspect, i.e. the capability of determining before executing the system if it exhibits some particular behaviour, is still matter of an intense research effort.

In this paper we investigate the a-priori verification of bottom-up build systems from the behavioural viewpoint, where a choreography is not known at the beginning of the developing process, but rather it is verified only later. We focus on the problem of deciding if, given a set of services, there can be some fruitful interaction among them; if yes, we focus also on the problem of determining such interaction. Our approach is based on specifying the services by means of the ConDec declarative language, and by exploiting its translation to the SCIFF Framework to automatically perform the verification task.

Index Terms—ConDec Service Modeling, Declarative Languages, A-priori Verification, Logic Programming.

I. INTRODUCTION

SERVICE ORIENTED COMPUTING emerged recently as an architectural paradigm for modeling and implementing business collaboration within and across organizational boundaries. The Web Service technology, currently the most advanced implementation of Service Oriented Architecture (SOA) principles, is almost established as the standard technology for current business implementations, thanks to the support it has received from the academics as well as from the industrial partners.

A key aspect in the success of Service Oriented Computing (SOC) is the possibility of composing different, heterogeneous services, yet achieving a complex system starting from more simple components. Interoperability at the level of data exchange, as well as at the level of service location and invocation, has been guaranteed by standards like WSDL [6].

Industrial tools are becoming available to support also the composition process, too. Although languages for defining composition rules and models have been proposed, but none of them has enjoyed the maturity level of the other standards. Initial proposals like BPMN [15], WS-CDL [7] and BPEL [3] have been criticized for their intrinsic procedural nature, while the need for open, declarative approaches has been recognized only later [4], [14].

Automatic verification of properties regarding the behavioural aspects of the composed systems is deemed as a crucial step, but a comprehensive solution is lacking. Currently,

DEIS - Department of Electronics, Informatics and Systems, University of Bologna. name.surname@unibo.it

the task of ensuring that two services can successfully cooperate is demanded to the software architect that is designing the system. Analogously, ensuring that the composed system will exhibit certain properties is a task that directly burden the developer. Although such guarantees can be verified by human users with small systems, there are serious doubts of achieving such results when the composed systems grow in dimension and interaction complexity. Hence, the task of automatically verifying a service composition a-priori (during the designing phase), is of the fundamental importance to foster the service composition and the “off-the-shelf” composition model .

Several approaches have been adopted to address the verification of service composition. A very common way consist of checking one service against a global description of a system, like in [1], [8]. In order to succeed, the following assumptions are usually made: 1) there is a description of the whole system, from a global point of view; 2) there is a description of the service under testing, such description not necessarily matching with the service internals; and 3) all services except the one under testing will behave as prescribed by the global specification (hence the global description can be used to reason upon the other services behaviour). Given this setting, the verification task determines if the behaviour of the service under testing is compatible with the global description (also named *choreography*). The choreography is intended as a sort of “legal, tight contract”, and plays a double role: it specifies the “boundaries” for the service under testing, and provides the expected behaviour of the other (unknown) peers. The obvious advantage of such approach is that the verification task involves only a service description and a choreography: the component “certified” as compliant can be then adopted to play a certain role within the global system, independently of the other peers.

In this paper we investigate the verification of service composition from another viewpoint. We start from the assumption that, at least in the beginning steps, a global choreography is not defined (or is not yet available). Beside a “top-down” developing method, where the developer starts designing the choreography and proceeds to refine the components in several steps, there is also a common “bottom-up” projecting style, where the developer simply starts to build up her application by putting together the already available components. If this is the case, the models of all parties (called *local models* thereafter) are directly composed, in order to make them interact and mutually benefit from each other, as in Figure 1a.

The first problem we try to address is: *given a composition of services, does exist a successful interaction? If yes, how is made such interaction?* In case of a positive answer to the

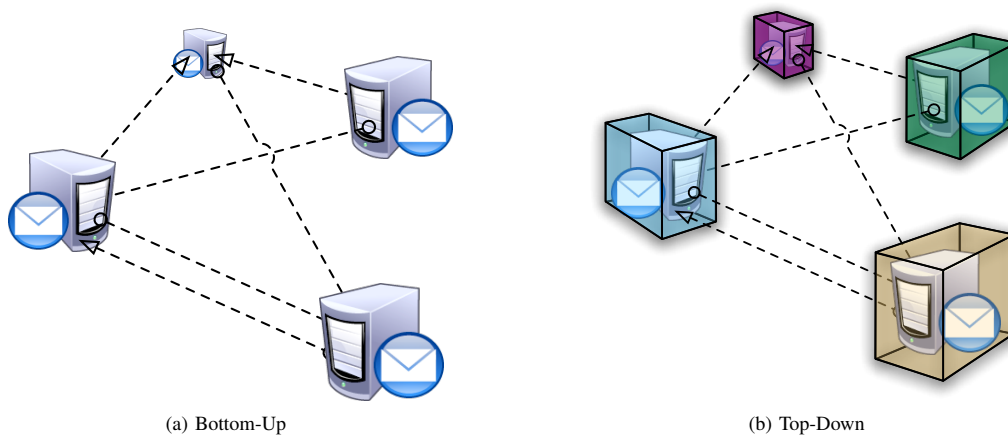


Fig. 1. Developing complex systems using simpler services.

first question, we say that the models are *compatible*. Beside the yes/no answer, we deem as fundamental also knowing as much as possible about such interactions. E.g., knowing in advance the supported interaction traces would help in the analysis of the global system features and properties. Of course, “compatible” does not mean that a successful interaction will be effectively achieved at run-time.

A second problem we discuss in this work is about choreographies, intended no more as a tight contract to be respected, but rather as a set of constraints representing the desired properties of the system. In this view, choreographies are not intended as the set of requirements each service should fulfill to interact, but rather a set of desired features that the global system will exhibit. Roughly, this problem can be formulated as follow: *given a compatible set of services, does exists some successful interaction that honors the choreography constraints? If yes, how is made such interaction?* Also in this case we aim to know not only a yes/no answer, but also some sort of fully/partially specified interaction trace.

In our approach, services and choreographies are represented by means of a declarative language, and in particular using the ConDec language [13]. We agree with the critics moved to procedural approaches in [4], [14]. In particular, choreographies (intended as a set of properties or constraints of the final resulting system) are more naturally represented in terms of declarative rules/constraints, rather than by sentences of a procedural language. In this work, we extend the original ConDec model to the concepts of roles, and provide definition of service composition compatibility on the resulting ConDec models.

The ConDec semantics originally proposed by the authors is given as Linear Temporal Logic (LTL) formulas. Recently, a further semantics in terms of the SCIFF Framework [2] has been provided to ConDec [11]. Another contribution of this work is the definition of a method for automatically perform the verification tasks, by exploiting the SCIFF-based semantics, and its proof procedure.

In Section II we briefly introduce the ConDec language, aimed to declarative describe open processes/services. Then in

Section III and in Section IV we try to better capture the notion of compatible services (compatible models of services) and of compliance to a choreography. In Section V we show how such properties are automatically verified exploiting the ConDec Language and its translation into the SCIFF Framework [2]. Finally we conclude and discuss future works.

II. MODELING A SERVICE BY MEANS OF THE CONDEC LANGUAGE

ConDec is a declarative, graphical language proposed by van der Aalst and Pesic [13] within the research field of Business Process Management (BPM). It aims to support specification, enactment and monitoring of Business Process, by means of constraints among activity executions. Constraints are declaratively expressed, as the authors claim that the adoption of a declarative approach fits better with complex, unpredictable processes, where a good balance between support and flexibility must be found. Although it has been originally proposed in the BPM context, it has been applied also in the far broader field of SOA. Former applications of ConDec regarded choreographies specification; in this paper, we adopt it to represent also service local models.

A. The ConDec Language

A ConDec model mainly consists of two parts: a set of activities, representing atomic units of work, and a set of relationships which constrain the way activities can be executed, and are therefore referred to as *constraints*. Constraints can be interpreted as policies/business rules, and may reflect different kinds of knowledge: external regulations and norms, internal policies and best practices, service/choreography goals, and so on.

Differently from procedural specifications, in which activities can be inter-connected only by means of control-flow relationships (sequence patterns, mixed with constructs supporting the splitting/merging of control flows), the ConDec language provides a number of control-independent abstractions to constrain activities, alongside the more traditional ones. In ConDec it is possible to insert past-oriented constraints, as

well as constraints that do not impose any ordering among activities.

Furthermore, while procedural specifications are closed, i.e., all what is not explicitly modeled is forbidden, ConDec models are *open*: activities can be freely executed, unless they are subject to constraints. This choice has two implications. First, a ConDec model accommodates many different possible executions, improving flexibility. Second, the language provides abstractions to explicitly capture not only what is mandatory, but also what is forbidden. In this way, the set of possible executions does not need to be expressed extensionally and models remain compact.

ConDec models do not impose a rigid scheduling of activities; instead, they leave the services free to execute activities in a flexible way, but respecting at the same time the imposed constraints. An execution trace, i.e. the set of the executed activities, we say that it is *supported* by a ConDec model if and only if it complies with all its constraints. Finally, it is important to note that well-defined ConDec models support only *finite* execution traces, because it must always be guaranteed that a BP will eventually terminate.

ConDec has been mapped to two different underlying logic frameworks, providing two different semantics for the ConDec constraints. Beside the originally proposed LTL mapping, in [11] a mapping to the SCIFF Framework has been proposed. SCIFF allows to define constraints in terms of the *happening* of events and *expectations* about the happening or the non-happening of other events. Events can be partially specified, by means of unbound variables. Possibly, such variables can be further constrained, a la CLP. E.g., it is possible to say that if a certain event happens at time T , then another event is expected to happen at a time T' with the CLP constraint $T' > T$.

A simple ConDec model where activities **a** and **b** can be executed many times, but the execution of one automatically exclude the execution of the other, can be expressed in SCIFF by means of two rules (Integrity Constraints, using the SCIFF terminology): $\mathbf{H}(a, T) \Rightarrow \mathbf{EN}(b, T')$ and $\mathbf{H}(b, T) \Rightarrow \mathbf{EN}(a, T')$ ¹. Note that such a simple model, if expressed using some procedural flow language such as for example Petri Nets, would lead to additional assumptions and choice points, thus making the final model pointlessly complicated.

Formally, a ConDec model \mathcal{CM} is composed by a set of activities, which represent *atomic* units of work (i.e., units of work associated to single time points, and relations among activities, used to specify constraints on their execution. Optional constraints are also supported, to express preferable ways to interact, but allowing the possibility to violate them.

Definition 1 (ConDec model \mathcal{CM}). A ConDec model is a triple $\langle \mathcal{A}, \mathcal{C}_m, \mathcal{C}_o \rangle$, where:

- \mathcal{A} is a set of *activities*, represented as boxes containing their name;
- \mathcal{C}_m is a set of *mandatory* constraints;
- \mathcal{C}_o is a set of *optional* constraints.

Given a ConDec model \mathcal{CM} , notations $\mathcal{A}^{\mathcal{CM}}$, $\mathcal{C}_m^{\mathcal{CM}}$ and $\mathcal{C}_o^{\mathcal{CM}}$ respectively denote the set of activities, mandatory and

¹The two rules state that if **a** Happens, then **b** is Expected Not to happen, and viceversa.

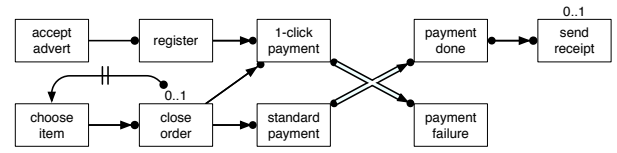


Fig. 2. A ConDec model.

optional constraints of \mathcal{CM} .□

B. A ConDec Example

Figure 2 shows the ConDec specification of a payment services. Boxes represent instances of activities. Numbers (e.g., 0; N..M) above the boxes are cardinality constraints that tell how many instances of the activity have to be done (e.g., never; between N and M). Edges and arrows represent relations(constraints) between activities. Double line arrows indicate alternate execution (after A , B must be done before A can be done again), while barred arrows and lines indicate negative relations (doing A disallows doing B). Finally, a solid circle on one end of an edge indicates which activity activates the relation associated with the edge. For instance, the execution of **accept advert** in Figure 2 does not activate any relation, because there is no circle on its end (a valid model could contain an instance of **accept advert** and nothing else); activity **register** instead activates a relation with **accept advert** (a model is not valid if it contains only **register**). If there is more than one circle, the relation is activated by each one of the activities that have a circle. Arrows with multiple sources and/or destinations indicate temporal relations activated/satisfied by either of the source/destination activities. The parties involved—a merchant, a customer, and a banking service to handle the payment—are left implicit.

In our example, the six left-most boxes are customer actions, **payment done/ payment failure** model a banking service notification about the termination status of the **payment** action, and **send receipt** is a merchant action. If **register** is done (once or more than once), then also **accept advert** must be done (before or after **register**) at least once. No temporal ordering is implied by such a relation. Conversely, the arrow from **choose item** to **close order** indicates that, if **close order** is done, **choose item** must be done at least once before **close order**. However, due to the barred arrow, **close order** cannot be followed by (any instance of) **choose item**. The 0..1 cardinality constraints say that **close order** and **send receipt** can be done at most once. 1-click payment must be preceded by **register** and by **close order**, whereas **standard payment** needs to be preceded only by **close order** (registration is not required). After 1-click or **standard payment**, either **payment done** or **payment failure** must follow, and no other **payment** can be done, before either of **payment done/failure** is done. After **payment done** there must be at most one instance of **send receipt** and before **send receipt** there must be at least a **payment done**. Sample valid models are: the empty model (no activity executed), a model containing one instance of **accept advert** and nothing else, and a model containing 5 instances of **choose item** followed

by a close order. A model containing only one instance of 1-click payment instead is not valid.

III. COMPATIBILITY AND LEGAL COMPOSITION

In this section we address the problem of establishing if some local models are *compatible* [12], i.e. if there exists an interaction trace allowed by the composed system. We first try to establish if a single model does indeed support at least one interaction trace (i.e., it is *conflict-free*, and then we extend the notion to the composed system. Note that all the following definitions are based on the idea of execution traces, i.e. on a set of events (ground facts), happened at certain time point.

First of all, we introduce the notion of \exists -entailment: the aim is to define somehow when a model guarantees a property. To do so, we look at the traces allowed by the local ConDec model, and verify such property directly on the allowed traces. Once we move to verify a property on a trace, the semantic of the entailment symbol could be referred to the SCIFF semantic (as we do), as well as to the LTL semantics.

Definition 2 (\exists -entailment). A property Ψ is \exists -entailed by a ConDec model \mathcal{CM} ($\mathcal{CM} \models_{\exists} \Psi$) if at least one execution trace supported by \mathcal{CM} entails the property as well. If that is the case, then one of the supported execution traces can be interpreted as an *example* which proves the entailment.

Definition 3 (Conflict-free model [12]). A ConDec model \mathcal{CM} is *conflict-free* iff it supports at least one possible execution trace, i.e., iff

$$\mathcal{CM} \models_{\exists} \text{true}$$

A conflicting model is an over-constrained model: it is impossible to satisfy all its mandatory constraints at the same time.

Then we generalize the idea of conflict-freedom to the composed model:

Definition 4 (Composite model [12]). Given n ConDec models $\mathcal{CM}^i = \langle \mathcal{A}^i, \mathcal{C}_m^i, \mathcal{C}_o^i \rangle$, the *composite model* obtained by combining $\mathcal{CM}^1, \dots, \mathcal{CM}^n$ is defined as :

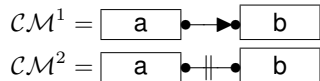
$$\text{COMP}(\mathcal{CM}^1, \dots, \mathcal{CM}^n) \triangleq \langle \bigcup_{i=1}^n \mathcal{A}^i, \bigcup_{i=1}^n \mathcal{C}_m^i, \bigcup_{i=1}^n \mathcal{C}_o^i \rangle$$

Definition 5 (Compatibility). Two ConDec models \mathcal{CM}^1 and \mathcal{CM}^2 are *compatible* if their composition is conflict-free, i.e., iff:

$$\text{COMP}(\mathcal{CM}^1, \mathcal{CM}^2) \models_{\exists} \text{true}$$

Obviously, the notion of compatibility can be generalized to the case of n local models. The detection of incompatibility means that a sub-set of the n local models leads to a conflict. note that checking compatibility could not be enough, as pointed out by the following example:

Example 1 (Trivial compatibility). Two local models



have been composed. The two models are compatible, because they both support the empty execution trace; therefore, by carrying out solely a compatibility check would seem that a composition can be actually built. However, as soon as an activity is executed, \mathcal{CM}^1 and \mathcal{CM}^2 are contradictory: both activity **a** and activity **b** can not be executed in the composite model. In the general case, if none of the local models contains constraints which impose the execution of a certain activity (i.e., `existenceN`, `exactlyN` and `choice` constraints), compatibility always returns a positive answer, because the empty execution trace is supported. \square

A. From Openness to Semi-Openness

Since a ConDec model is open, it implicitly allows the execution of activities not explicitly contained in the model itself. The following example clarifies the point. This characteristic could cause undesired compositions to be evaluated as correct, as in the following example.

Example 2 (Composition and openness issues). A customer wants to find a seller to interact with. The customer comes with a ConDec model representing its own desired constraints and requirements. In particular, they express that:

- the customer wants to receive a good from a seller;
- if the customer pays for a good, then she expects that the seller will deliver it;
- before paying, the customer wants the seller to provide a guarantee that the payment method is secure.

Figure 3 shows the ConDec graphical models (\mathcal{CM}_C) of the customer and of three candidate sellers. The three sellers differ for what concerns the possibility of emitting a guarantee upon request:

- 1) the seller depicted in Figure 3(b) (\mathcal{CM}_S^1) explicitly states that it does not provide any guarantee upon request;
- 2) the seller depicted in Figure 3(c) (\mathcal{CM}_S^2) explicitly supports the possibility of providing a guarantee;
- 3) the seller depicted in Figure 3(d) (\mathcal{CM}_S^3) does not mention provide guarantee among its activities.

Following Definition 5 checking compatibility between \mathcal{CM}_C and the three candidate sellers would state that \mathcal{CM}_C is not compatible with \mathcal{CM}_S^1 , but it is compatible with \mathcal{CM}_S^2 and \mathcal{CM}_S^3 . In particular, the two compositions $\mathcal{CM}_C \cup \mathcal{CM}_S^2$ and $\mathcal{CM}_C \cup \mathcal{CM}_S^3$ produce exactly the same global model. However, while the answer given for the first two compositions is in accordance with the intuitive notion of compatibility, the third one is not. In fact, when \mathcal{CM}_C is composed with \mathcal{CM}_S^2 , the behaviour of the seller is modified in that also the constraints of the customer must be respected. Contrariwise, when the composition between \mathcal{CM}_C and \mathcal{CM}_S^3 is established, the local model of the customer has the effect of changing the local model of the seller, augmenting it with a new **provide guarantee** activity. During the execution, the customer would expect to receive a guarantee before paying, but this capability has not been mentioned by the seller in its local model, and therefore there could be the case that it is not supported. \square

The example clearly shows that the openness assumption must be properly revised when dealing with the composition

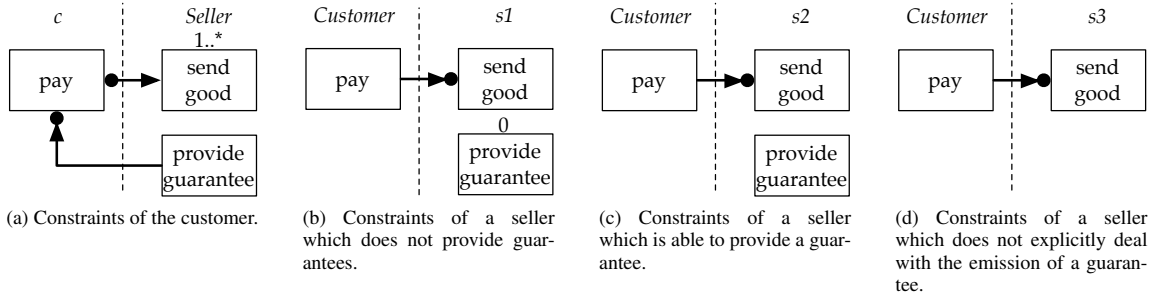


Fig. 3. Local models of a customer and of three candidate sellers.

problem. To ensure that a composition can be established, the obtained global model must obey to the following *semi-openness* requirement: for each involved party, *the activities under the responsibility of that party must also explicitly appear in its local model*.

B. Augmenting ConDec Models with Roles and Participants

In order to ensure the *semi-openness* assumption, each activity must be associated to its corresponding originator or role. The following definition extends the basic definition of a ConDec model with such a relationship.

Definition 6 (Augmented ConDec model). An augmented ConDec model is a 4-tuple $\langle \mathcal{AO}, \mathcal{AR}, \mathcal{C}_m, \mathcal{C}_o \rangle$, where:

- \mathcal{AO} is a set of (A, O) pairs where A is an activity and O is its originator;
- \mathcal{AR} is a set of (A, R) pairs where A is an activity and R represents the role of its originator;
- \mathcal{C}_m is a set of mandatory constraints over \mathcal{AO} and \mathcal{AR} ;
- \mathcal{C}_o is a set of optional constraints over \mathcal{AO} and \mathcal{AR} .

If $\mathcal{AR} = \emptyset$, the model is *completely grounded*. Contrariwise, if $\mathcal{AO} = \emptyset$ the model is *abstract*.

In this respect, a ConDec local model is defined as an augmented model containing also an indication about the identifier of the local model, and where an activity is associated either to such an identifier, or to an abstract role.

Definition 7 (Local augmented model). A ConDec local augmented model is a 5-tuple $\langle ID, \mathcal{AO}, \mathcal{AR}, \mathcal{C}_m, \mathcal{C}_o \rangle$, where:

- ID is the identifier of the participant executing the local model;
- the other elements retain the meaning of Definition 6;
- \mathcal{AO} is a set containing only elements of the type (A, ID) .

A role identifies a *class* of originators; in the composition process, abstract roles employed in each local model are mutually grounded to concrete local models which participate to the composition.

Definition 8 (Grounding of a model). Given an augmented model $\mathcal{C}Maug = \langle \mathcal{AO}, \mathcal{AR}, \mathcal{C}_m, \mathcal{C}_o \rangle$ and a function plays mapping roles to concrete identifiers (i.e., stating that a certain identifier “plays” a given role), the grounding of $\mathcal{C}Maug$ on plays is obtained by substituting each role R_i with the corresponding concrete participant identifier $\text{plays}(R_i)$:

- $\mathcal{AO} \downarrow_{\text{plays}} \triangleq \mathcal{AO} \cup \{(A, \text{plays}(R_i)) \mid R_i \in \text{dom}(\text{plays}) \wedge (A, R_i) \in \mathcal{AR}\}$;
- $\mathcal{AR} \downarrow_{\text{plays}} \triangleq \mathcal{AR} / \{(A, R_i) \mid R_i \in \text{dom}(\text{plays})\}$;
- $\mathcal{C}_m \downarrow_{\text{plays}}$ and $\mathcal{C}_o \downarrow_{\text{plays}}$ are updated accordingly.

If $\mathcal{AR} \downarrow_{\text{plays}} = \emptyset$, each role has been substituted by a concrete identifier and the model becomes ground. A legal composite ConDec can be now characterized as an augmented model obtained by composing a set of local models, each one grounded by taking into account the other ones, s.t. the composition is ground.

Definition 9 (Augmented composite model). Given a set of augmented local models $\mathcal{L}^i = \langle ID_i, \mathcal{AO}^i, \mathcal{AR}^i, \mathcal{C}_m^i, \mathcal{C}_o^i \rangle$ ($i = 1, \dots, n$) and a function plays mapping roles to identifiers, the composition of the local models w.r.t. plays is defined as

$$\text{COMP}(\mathcal{L}^1, \dots, \mathcal{L}^n)_{\text{plays}} = \bigcup_{i=1}^n \mathcal{L}^i \downarrow_{\text{plays}}$$

where the union of two augmented models is a shortcut representing the union of each corresponding element. A composition is *legal* iff $\text{COMP}(\mathcal{L}^1, \dots, \mathcal{L}^n)_{\text{plays}}$ is ground (see Definition 6).

It is now possible to revise the notion of compatibility reflecting also the semi-openness assumption.

Definition 10 (Strong compatibility). n local models $\mathcal{L}^i = \langle ID_i, \mathcal{AO}^i, \mathcal{AR}^i, \mathcal{C}_m^i, \mathcal{C}_o^i \rangle$ ($i = 1, \dots, n$) are *strong compatible* under plays iff their augmented composition $\text{COMP}(\mathcal{L}^1, \dots, \mathcal{L}^n)_{\text{plays}} = \langle \mathcal{AO}^{\cup}, \mathcal{AR}^{\cup}, \mathcal{C}_m^{\cup}, \mathcal{C}_o^{\cup} \rangle$ satisfies the following properties:

- $\text{COMP}(\mathcal{L}^1, \dots, \mathcal{L}^n)_{\text{plays}}$ is legal;
- $\text{COMP}(\mathcal{L}^1, \dots, \mathcal{L}^n)_{\text{plays}}$ is conflict-free;
- for each (a, ID_i) which belongs to \mathcal{AO}^{\cup} but does not belong to \mathcal{AO}^i , it must hold that:

$$\text{COMP}(\mathcal{L}^1, \dots, \mathcal{L}^n)_{\text{plays}} \models_{\forall} \text{absence}((a, ID_i))$$

The third point states that if a certain activity a has been associated to a participant ID_i , but ID_i has not explicitly mentioned a in its specification, then the composition must always ensure that a cannot be executed.

Example 3. Let us re-examine the compatibility between the local models of the customer and the second seller shown in Figure 3, supposing that their identifiers are respectively *alice*

and *hutter*, and *customer* and *seller* represent their roles. In the composition, *alice* plays the role of *customer*, and *hutter* plays the role of *seller*. Hence, $\text{plays}(\text{alice}) = \text{customer}$ and $\text{plays}(\text{hutter}) = \text{seller}$.

By adopting the definition of augmented models, the ConDec diagram of *alice* is:

$$\mathcal{L}_{\text{alice}} = \{ \{ (\text{pay}, \text{alice}) \}, \\ \{ (\text{send good}, \text{seller}), (\text{provide guarantee}, \text{seller}) \}, \\ \{ \text{existenceN}(1, (\text{send good}, \text{seller})), \dots \}, \\ \emptyset \}$$

The grounding of *alice* w.r.t. the plays function is $\mathcal{L}_{\text{alice}} \downarrow_{\text{plays}} =$

$$\{ \{ (\text{pay}, \text{alice}), (\text{send good}, \text{hutter}), (\text{provide guarantee}, \text{hutter}) \}, \\ \emptyset, \\ \{ \text{existenceN}(1, (\text{send good}, \text{hutter})), \dots \}, \\ \emptyset \}$$

The grounding of *hutter* is obtained in a similar way.

When the two local models are composed, the grounding of *alice* causes $(\text{provide guarantee}, \text{hutter})$ to belong to the set \mathcal{AO}^{\cup} of the composition. Since the execution trace $\text{provide guarantee} \rightarrow \text{pay} \rightarrow \text{send good}$ is compliant with the composition but $(\text{provide guarantee}, \text{hutter}) \notin \mathcal{AO}^{\text{hutter}}$, the two local models are not strong compatible. \square

IV. CONFORMANCE TO A CHOREOGRAPHY

Once a global model has been obtained through the composition step, and after a strong compatibility property has been verified, the application developer can move to further analyse the resulting system, trying to understand if it entails some desired properties.

We represent a choreography as an augmented abstract ConDec model (i.e., an augmented model associating all the activities to roles and not to concrete participants – see Definition 6). When realizing a choreography with a set of concrete local models, different possible errors may arise:

- *Conflicting composition*: independently from the choreography, the chosen local models are not compatible.
- *Local non-conformance*: a concrete local model is not able to correctly play, within the choreography, the role it has been assigned to.
- *Global non-conformance*: even if each single local model is able to correctly play the role it has been assigned to, it could be the case that the global obtained model does not conform the choreography anymore.

It could be also the case that a participant would not be able to play the role it has been assigned to, but it would anyway be able to take part to a conforming composition. Such a situation may arise because when the constraints of each local model are joint with the ones of the others, the constraints of the participant could be correctly “completed”.

As a consequence it is necessary to first check that the composition is conflict-free, and then verify the whole composition against the choreography. To verify that a composition conforms to a desired choreography, two approaches can be followed. The *weak* approach states that the composition must be consistent with the choreography constraints in at least one

supported execution, while the *strong* approach requires to guarantee that any execution supported by the composition respects the choreography.

Definition 11 (Weak conformance). A composition of local models $\text{COMP}(\mathcal{L}_1, \dots, \mathcal{L}_n)_{\text{plays}}$ is *weak conformant* with a choreography *Chor* iff:

- $\mathcal{L}_1, \dots, \mathcal{L}_n$ are strong compatible w.r.t. plays (see Definition 10);
- $\text{COMP}(\mathcal{L}_1, \dots, \mathcal{L}_n)_{\text{plays}} \models_{\exists} \text{Chor} \downarrow_{\text{plays}}$.

Definition 12 (Strong conformance). A composition of local models $\text{COMP}(\mathcal{L}_1, \dots, \mathcal{L}_n)_{\text{plays}}$ is *strong conformant* with a choreography *Chor* iff:

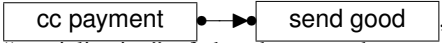
- $\mathcal{L}_1, \dots, \mathcal{L}_n$ are strong compatible w.r.t. plays (see Definition 10);
- $\text{COMP}(\mathcal{L}_1, \dots, \mathcal{L}_n)_{\text{plays}} \models_{\forall} \text{Chor} \downarrow_{\text{plays}}$.

Example 4 (Weak and strong conformance). Let us consider a simple (fragment of a) choreography involving two roles – a customer and a seller. The choreography states that:

- 1) two possible payment methods are available to the customer (payment by credit card and payment by cash);
- 2) the customer can pay only after having closed the order;
- 3) if the customer pays, then the seller is entitled to send the ordered good and, conversely, a good is sent to the customer only if a payment has been previously done.

Figure 4 shows the ConDec model of the resulting choreography, and three possible local models which can be composed to realize such a choreography. In particular, *alice* can play the role of *Customer*, while *hutter* and *lewis* can play the role of *Seller*.

Let us first consider the composition obtained by combining the model of *alice* with the one of *lewis*. The composition is strong conformant with the choreography:

- The choreography allows an open choice on the payment modality, and both local models only deal with payment by credit card.
- The combination of the constraints which relate the payment with the delivery of the good in the two local models leads to obtain the following constraint , which is a “specialization” of the choreography one (no choice is present).
- *alice* states that before paying, she wants to close the order, and that between two payments at least one **close order** must be executed; such a constraint is a specialization of the simple **precedence** constraint contained in the choreography.

The composition obtained by combining the model of *alice* with the one of *hutter* is instead not strong conformant. In fact, *hutter* does not impose any temporal ordering between the payment and the delivery of the good. Therefore, it could be possible that the good is sent twice: one time before the payment of *alice*, and another time afterwards. I.e., the following execution trace is supported by the composition: **close order** \rightarrow **send good** \rightarrow **cc payment** \rightarrow **send good**. The first execution of the **send good** activity is not preceded by a payment, thus violating a prescription of the choreography.

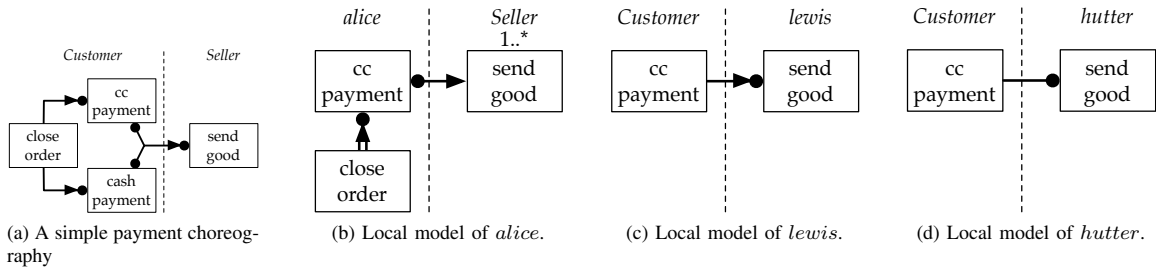


Fig. 4. A simple choreography and three candidate local models (one customer and two sellers).

However, the composition is weak conformant, because it supports different possible executions which comply with the choreography. \square

V. VERIFICATION THROUGH SCIFF

The SCIFF framework has been originally developed for the declarative specification and run-time verification of interaction protocols in the context of open Multi-Agent Systems [2]. It features:

- A rule-based language for modeling all the constraints that must be respected by the events characterizing the executions of the system under study. These rules relate the concepts of event occurrence with the one of expected/forbidden event, to model the (un)desired courses of interaction when a given situation is reached during the interaction. Events are modeled as logic programming terms (possibly containing variables), and are associated to an explicit execution/expected time; times and variables can be constrained by means of CLP constraints and Prolog predicates.
- A clear declarative semantics characterizing the execution traces compliant with the modeled rules.
- A corresponding proof procedure, sound and complete w.r.t. the declarative semantics, which is able to dynamically acquire the events occurring during a specific execution of the system, and check on-the-fly their compliance with the modeled rules.

In the last years, the framework has been applied in other contexts, such as clinical guidelines, business contracts and processes, service choreographies [9]. In particular, a complete mapping of all the ConDec constraints in terms of SCIFF rules has been provided, proving its soundness w.r.t. the original ConDec semantics (specified by means of LTL formulae) and discussing its impact on the verification techniques.

The SCIFF proof procedure has been then extended to deal also with the static verification of interaction protocols. g-SCIFF is the generative variant of SCIFF devoted to this task: instead of checking if a given (partial or complete) execution trace is compliant with the modeled rules, it is able to generate compliant execution traces or to return a negative answer if none exists [10]. In this way, g-SCIFF can be suitably exploited to effectively prove whether a ConDec model (translated to SCIFF) \models_{\exists} or \models_{\forall} a given property [9]. In particular, a ConDec model \models_{\exists} a given property if and only if the g-SCIFF proof procedure is able to generate at

least one execution trace compliant with the model and the property; such an execution trace can be considered as an example proving the existential entailment of the property. The case of \models_{\forall} is reduced, similarly to model checking, to the \models_{\exists} of the complemented property; the generation of an execution trace compliant with the model and the completed property can be considered as a counter-example showing that the original property is not entailed by the model in all its supported executions.

A. Compatibility Verification with g-SCIFF

Let us briefly describe how g-SCIFF carries out the compatibility verification between the customer's model shown in Figure 3(a) and each one of the three sellers modeled in Figures 3(b)-(d). As we have seen, the constraints of a composite model are obtained by joining all the constraints of the local models. Let us denote the composition of the customer's model with each seller's model with respectively $\mathcal{CM}_c^{s_1}$, $\mathcal{CM}_c^{s_2}$ and $\mathcal{CM}_c^{s_3}$. The first step, according to Definition 10, is to verify whether the composite model is legal; this is a syntactic test which can be trivially proven for all the three composite models: the role of *Customer* is grounded on c , and the role of *Seller* is respectively grounded on s_1 , s_2 and s_3 . The second and third step require instead the presence of a verifier able to prove conflict-freeness (\models_{\exists}) and to check if the composite model meet the semi-openness requirement (\models_{\forall}). When verifying the conflict-freeness of $\mathcal{CM}_c^{s_2}$ and $\mathcal{CM}_c^{s_3}$, g-SCIFF operates as follows:

- it starts from the $1..*$ constraint on `send good`, generating an occurrence of the event;
- this generated occurrence triggers the precedence ($\longrightarrow\bullet$) constraint involving `send good` and `pay`, leading to generate a previous payment;
- the generated payment, in turn, triggers the precedence ($\longrightarrow\bullet$) constraint involving `pay` and `provide guarantee`, leading to generate a previous emission of a guarantee.

At the end of the verification process, the following sample execution trace is therefore produced by g-SCIFF: `provide guarantee` \rightarrow `pay` \rightarrow `send good`². When verifying the compatibility of $\mathcal{CM}_c^{s_1}$, instead, after the third step g-SCIFF realizes that the execution of `provide guarantee` clashes with

²The temporal relationships imposing the orderings between the three generated events are represented with CLP constraints

the 0 cardinality constraint (absence constraint) imposed by s_1 , and returns a negative answer attesting the incompatibility of the local models.

The last requirement to be verified is that for each (a, ID_i) which belongs to $\mathcal{CM}_c^{s_2}/\mathcal{CM}_c^{s_3}$ but does not belong to the corresponding local model, it must hold that the absence of a is \models_{\forall} by the composite model. In the case of $\mathcal{CM}_c^{s_2}$, no such activity actually exists, and therefore the requirement is directly met, attesting that the two local models are indeed compatible. Contrariwise, $\mathcal{CM}_c^{s_3}$ contains the activity (provide guarantee, s_3) which is however not contained in the local model of s_3 . Therefore, g-SCIFF must prove whether all execution traces compliant with the composite model do not contain the execution of the provide guarantee activity. As already pointed out, \models_{\forall} is reduced to \models_{\exists} by complementing the property; in this specific case, the verification reduces to check whether at least one execution trace compliant with the composite model exists s.t. at least one execution of provide guarantee is contained in the trace. The execution trace provide guarantee \rightarrow pay \rightarrow send good, produced by g-SCIFF when checking the conflict-freedom of $\mathcal{CM}_c^{s_3}$, does satisfy the complemented property, and can be therefore considered as a counter-example showing that the semi-openness requirement is not met by the composite model, i.e., that c and s_3 are not compatible.

Finally, note that the conformance verification of a composite model with a choreography is carried out by g-SCIFF similarly to the case of compatibility. In the case of strong conformance, each constraint involved in the choreography is complemented and then separately checked w.r.t. \models_{\exists} . If at least one complemented property is \models_{\exists} , then the composition is not strong conforming with the choreography.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have address some issues related to the process of composing complex system by using existing components, in the context of the Service Oriented Architectures. Our approach hypothesizes a bottom-up process in composing the global system, where the components are put together in order to achieve some interaction, and choreography constraints are taken into account only at a second stage. Peculiarities of our solution are 1) the use of a open, declarative, logic based language to represent models of the services and also the choreographies; 2) the exploit of the SCIFF Proof Procedure, and in particular of the g-SCIFF proof, to automatically perform all the verification tasks; and 3) beside a yes/no answer, our approach provides as output also some interaction trace that can be used to reason upon and analyse the global system.

This work is still in its preliminary stage, although some successful experimental results have been already obtained. Future works will be devoted to better assess the theory behind the solution, and to provide a better comparison with other approaches available in the literature, like [5], [8]. In particular, on the theme of the a-priori verification there is a huge research literature, as well as on the topic of reasoning on choreographies and roles. A further issue we intend to investigate is

related to the possibility of introducing “soft” ConDec constraints: currently, constraints are considered as hard, and not respecting one constraint leads to an incompatibility response. We are hypothesizing situations where behavioural interface specifications can also comprehend compensations actions and explicit management of violations.

ACKNOWLEDGMENT

This work has been partially supported by the FIRB project TOCAI.it (RBNE05BFRK) and by the Italian MIUR PRIN 2007 project No. 20077WWCR8.

REFERENCES

- [1] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and M. Montali. An Abductive Framework for A-Priori Verification of Web Services. In A. Bossi and M. J. Maher, editors, *Proceedings of the 8th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*, pages 39–50. ACM Press, 2006.
- [2] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Verifiable Agent Interaction in Abductive Logic Programming: the SCIFF framework. *ACM Transactions on Computational Logic*, 9(4), 2008.
- [3] T. Andrews, F. Curbera, H. Dholakia, Y. Golland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2003.
- [4] A. Barros, M. Dumas, and P. Oaks. A Critical Overview of the Web Services Choreography Description Language (WS-CDL). *BPTrends*, 2005.
- [5] A. K. Chopra and C. P. Singh. Producing Compliant Interactions: Conformance, Coverage, and Interoperability. In *4th International Workshop on Declarative Agent Languages and Technologies IV (DALT 2006), Selected, Revised and Invited Papers*, volume 4327 of *Lecture Notes in Computer Science*, pages 1–15. Springer Verlag, 2006.
- [6] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. <http://www.w3.org/TR/wsdl>, 2001.
- [7] N. Kavantzaz, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon. Web Services Choreography Description Language Version 1.0, 2004. <http://www.w3.org/TR/ws-cdl-10/>.
- [8] M. Baldoni and C. Baroglio and A. Martelli and V. Patti. A Priori Conformance Verification for Guaranteeing Interoperability in Open Environments. In A. Dan and W. Lamersdorf, editors, *Proceedings of the 4th International Conference on Service-Oriented Computing (ICSOCC 2006)*, volume 4294 of *Lecture Notes in Computer Science*. Springer Verlag, 2006.
- [9] M. Montali. *Specification and Verification of Declarative Open Interaction Models: a Logic-Based Framework*. PhD thesis, University of Bologna, 2009.
- [10] M. Montali, M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni. Verification from Declarative Specifications Using Logic Programming. In M. G. D. L. Banda and E. Pontelli, editors, *ICLP*, number 5366 in LNCS, pages 440–454. Springer Verlag, 2008.
- [11] M. Montali, M. Pesic, W. M. P. van der Aalst, F. Chesani, P. Mello, and S. Storari. Declarative Specification and Verification of Service Choreographies. *ACM Transactions on the Web - Accepted*, 2009.
- [12] M. Pesic. *Constraint-Based Workflow Management Systems: Shifting Controls to Users*. PhD thesis, Beta Research School for Operations Management and Logistics, Eindhoven, 2008.
- [13] M. Pesic and W. M. P. van der Aalst. A Declarative Approach for Flexible Business Processes Management. In *Proceedings of the BPM 2006 Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 169–180. Springer Verlag, 2006.
- [14] W. M. P. van der Aalst, M. Dumas, A. H. M. ter Hofstede, N. Russell, H. M. W. Verbeek, and P. Wohead. Life After BPEL? In M. Bravetti, L. Kloul, and G. Zavattaro, editors, *Proceedings of the 2nd International Workshop on Web Services and Formal Methods (WS-FM 2005)*, volume 3670 of *Lecture Notes in Computer Science*, pages 35–50. Springer Verlag, 2005.
- [15] S. A. White. Business Process Modeling Notation Specification 1.0. Technical report, OMG, 2006.

Web Services Synchronization in Composition Scenarios

Hamdi Yahyaoui¹, Zakaria Maamar², Jamal Bentahar³, and Khouloud Boukadi⁴

¹KFUPM, Dhahran, KSA — ²Zayed University, Dubai, U.A.E — ³Concordia University, Montreal, Canada—

⁴École des Mines, Saint-Etienne, France

Abstract

This paper discusses Web services synchronization at the composition level. Synchronization aims at assisting independent parties coordinate their actions and thus, avoid conflicts. Our previous work on synchronization primarily focused on the component level and shed the light on two types of behaviors related to specifying Web services. The control behavior defines the business logic that underpins the functioning of a Web service, and the operational behavior regulates the execution progress of this control behavior by stating the actions to carry out and the constraints to put on this progress. Control and operational behaviors continue to be used to specify composite Web services with respect to the orchestration schemas that these composite Web services have to comply with whether centralized or peer-to-peer. As a result, various types of messages to achieve synchronization are developed per type of orchestration schema. Experiments showing the use of these messages are reported in this paper as well.

Keywords. *Composition, Synchronization, Web service.*

1 Introduction

In [8] and [9], we investigated the synchronization issue of (“isolated”) Web services independently of the composition scenarios in which these Web services could take part. This investigation shed the light on two types of behaviors namely *control* and *operational* that were both used to specify and exhibit the functioning of Web services. The control behavior illustrates the business logic of the functionality, e.g., CarRental, of a Web service, while the operational behavior frames the progress of executing the business logic of this Web service at run time.

As the literature review points out [3], it is known that the “beauty” of Web services resides in their capacity to be composed into high-level business processes known as composite Web services. Composition is suitable for users’ requests that cannot be satisfied by any single, available Web service, whereas a composite Web service obtained by combining available Web services might be used. Com-

position design and development are bound to a specification that describes, at design time, multiple elements such as execution order of component Web services, data dependencies between component Web services, and corrective strategies in case component Web services raise exceptions. At run time, the composition specification is triggered, which means identifying and invoking component Web services, overseeing their execution, coordinating their actions, and initiating corrective strategies if needed. Different specifications related to Web services composition currently exist such as BPEL (*de facto* standard) and WSCI.

In term of execution, Web services composition can be structured along two types of orchestration [1]: *centralized* or *peer-to-peer* (P2P) (i.e., decentralized). On the one hand, centralized orchestration like its name hints relies on a centralized module (e.g., BPWS4J) that coordinates and tracks all the execution activities related to component Web services in terms of when to invoke them, what to expect out of their invocation, what data they exchange, how to pass on these data, just to cite a few. On the other hand, P2P orchestration excludes the centralized module and promotes direct interactions between component Web services. This makes Web services aware of some of their direct acquaintances during composition, which means the necessity of empowering these Web services with appropriate knowledge and mechanisms in order to support direct interactions. eFlow [2] is an example of Web services-based systems that adopts a centralized orchestration, whereas PCAP [10] is an example of Web services-based systems that adopts a P2P orchestration.

This paper extends the synchronization initiative we report in [9] by leveraging this time our research findings and thoughts from the component to the composition levels. In [5], we applied the separation between control and operational behaviors to model check orchestration-based composite Web services. The control behavior is used to extract the desired properties to be checked in the model of composite Web services captured by the operational behavior. In this paper, our primary objective is to address the following issues per type of orchestration: what synchronization mechanisms are required to set up, what messages im-

plement these mechanisms, how these messages are tracked during synchronization, how synchronization and execution are interleaved, and how the correctness of these messages is proved. In this extended work, Web services are no longer treated as isolated components but as integral components of composition scenarios. Analyzing the synchronization of Web services at the composition level offers some direct benefits. First of all, it would be possible to dissociate the behaviors of Web services at the composite level from the behaviors of these Web services at the component level. Second, it would be possible to track the interactions that occur between the Web services from the component to the composite levels and *vice-versa*. Finally, it would be possible to work out the necessary synchronization mechanisms per type of composition orchestration whether centralized or P2P.

Section 2 discusses the commonalities and differences between the component and composite levels and provides a running scenario. Section 3 reports on the synchronization work that was done at the component level. Section 4 discusses synchronization at the composite level with focus on the P2P schema. Prior to concluding in Section 6, some experimental details are given in Section 5.

2 Background

2.1 Component vs. composition levels

In a composition scenario, we classify interactions that involve composite and component Web services into vertical (from composite Web service to component Web service) and horizontal (from component Web service to another component Web service). By establishing an interaction session, the initiator of a message aims at making the recipient of this message behave and take actions according to the content of this message. In the following, we identify the acceptable actions that a message initiator can execute over a potential recipient during vertical and horizontal interactions. The objective of identifying these actions is to facilitate the definition of the relevant synchronization messages that would be suitable per type of interaction.

In vertical interactions, a centralized orchestration of Web services composition is implemented. Here, a composite Web service through the centralized module has the authority to carry out the following actions over a component Web service:

“*Invite*” action makes the composite Web service request the participation of the component Web service in its composition scenario¹;

“*Ping*” action makes the composite Web service check the liveness of the component Web service that accepted

its invitation of participation; there is no guarantee that the component Web service is still part of a composition scenario at time of invocation;

“*Trigger*” action makes the composite Web service initiate the execution of the component Web service;

“*Audit*” action makes the composite Web service monitor the performance of the component Web service for assessment purposes; service level agreements motivate the audit exercise;

And, “*retract&invite*” action makes the composite Web service withdraw the component Web service from its composition due to poor performance for example. This yields into searching for another replacement Web service that will be added to this composition.

In horizontal interactions, a P2P orchestration of Web services is implemented. Here, a component Web service has the authority to carry out the following actions over a peer:

“*Invite*” action makes the component Web service request the participation of the peer in the current composition scenario;

“*Ping*” action makes the component Web service check the liveness of the peer that accepted its invitation of participation; there is no guarantee that the peer is still part of a composition scenario at time of invocation;

And, “*trigger*” action makes the component Web service initiate the execution of the peer.

Compared to the vertical interactions in the centralized orchestration, “*audit*” and “*retract*” actions are excluded from the horizontal interactions in the P2P orchestration. Essentially, this is due to the challenges that are posed when tracking the performance of Web services and replacing them if needed. Not all providers would like to have their Web services audited by the Web services of other providers for reasons that could be related to security, privacy, competitiveness, etc. In a centralized orchestration, providers do not mutually interact with each other and might not even know that they are parts of the same composition scenario. The absence of “*audit*” and “*retract*” actions in a P2P orchestration sheds the light on the necessity of developing appropriate mechanisms that should take into account concerns like privacy and competitiveness. However, these mechanisms do not fall into the scope of this paper.

2.2 Running scenario

Our running scenario concerns a university student who is in the process of organizing a cookout party to celebrate his recent graduation. We identify hereafter the Web services along with their activities that will implement this party’s logistics.

CateringWS: searches for and contacts catering companies according to some criteria like allocated budget, num-

¹Component Web services invitation is discussed in [6].

ber of expected guests, type of cuisine, etc.

GuestWS: sends invitees invitations, keeps track of confirmed invitations, reminds late invitees for confirmation, etc.

PlaceBookingWS: looks for a place to host the cookout party, books the place, completes the necessary paperwork like payment, etc.

WeatherWS: checks weather forecast for the day of the cookout party. In case of bad weather, the party takes place at the student's place.

In our initial synchronization project [9], *state charts* were selected to specify component Web services independently of any composition scenario (Fig. 2 (a)). For the sake of compliance, we continue doing so when modeling the specification of composition scenarios. However states correspond this time to Web services taking part in these scenarios (Fig. 1).



Figure 1. Specification of the cookout party

3 Specification of Web services

3.1 Control and operational behaviors

The *control behavior* shows the business logic that underpins the functioning of a Web service with respect to its functionality. A business logic is domain-application dependent (e.g., healthcare) and changes from one case study to another according to various requirements such as user (e.g., minimum age to submit an application), security (e.g., type of encryption algorithm), etc.

The *operational behavior* guides the execution progress of the business logic of a Web service. To this end, this behavior relies on a specific number of states, which are activated, not-activated, done, aborted, suspended, and compensated. These states are reported in the field of transactional Web services [11] and common to a certain extent to all Web services (and to any software application) regardless of their functionalities, origins, and locations.

As mentioned in Section 1, the control and operational behaviors of a Web service are modeled using state charts. This exercise of modeling is hereafter interleaved with some formal definitions and illustrative examples.

Definition 1 (Web Service Behavior). The behavior of a Web service is a 5-tuple $\mathcal{B} = \langle \mathcal{S}, \mathcal{L}, \mathcal{T}, s^0, \mathcal{F} \rangle$ where: \mathcal{S} is a finite set of state names; $s^0 \in \mathcal{S}$ is the initial state; $\mathcal{F} \subseteq \mathcal{S}$ is a set of final states; \mathcal{L} is a set of labels; and $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$ is the transition relation. Each transition $t = (s^{src}, l, s^{tgt})$ consists of a source state $s^{src} \in \mathcal{S}$, a target state $s^{tgt} \in \mathcal{S}$, and a transition label $l \in \mathcal{L}$. From now

on, we qualify transitions in the behavior of a Web service as *intra-behavior*. \square

The control and operational behaviors of a Web service are defined as instances of the behavior of this Web service (Definition 1). These two behaviors are denoted by $\mathcal{B}_{co} = \langle \mathcal{S}_{co}, \mathcal{L}_{co}, \mathcal{T}_{co}, s_{co}^0, \mathcal{F}_{co} \rangle$ and $\mathcal{B}_{op} = \langle \mathcal{S}_{op}, \mathcal{L}_{op}, \mathcal{T}_{op}, s_{op}^0, \mathcal{F}_{op} \rangle$, respectively.

Example 1: Fig. 2 (a) is a state chart of the control behavior of *WeatherWS*. Several states like *city-located* (initial state), *report-delivered* (final state), and *search-canceled*, and several transitions like (*city-located*, *unavailable*, *search-canceled*) are included in this state chart. In this transition example, *city-located* and *search-canceled* are the source and target states, respectively, and *unavailable* is the transition's label.

Example 2: Fig. 2 (b) is another state chart that illustrates this time the operational behavior of *WeatherWS*. Similar to the control behavior, several states like *not-activated* and *suspended*, and transitions like (*compensated*, *rolling-back*, *not-activated*) and (*activated*, *failure*, *aborted*) are identified in this state chart.

In Fig. 2, the control and operational behaviors of a Web service include different finite sequences that connect states and transitions together. We refer to these sequences as *paths* and define them as follows:

Definition 2 (Path in Web Service Behavior). A path $p^{i \rightarrow j}$ in the behavior \mathcal{B} of a Web service is a finite sequence of states and transitions starting from state s^i and ending at state s^j and is denoted as follows: $p^{i \rightarrow j} = s^i \xrightarrow{l^i} s^{i+1} \xrightarrow{l^{i+1}} s^{i+2} \dots s^{j-1} \xrightarrow{l^{j-1}} s^j$ such that $\forall k \in \{i, j-1\} : (s^k, l^k, s^{k+1}) \in \mathcal{T}$ (exponents in state names are here given for notational purposes only). \square

Example 3: Let l^1 (resp. l^2) = *start* (resp. *commitment*) in Fig. 2 (b). *not-activated* $\xrightarrow{l^1}$ *activated* $\xrightarrow{l^2}$ *done* is a path in the operational behavior of *WeatherWS*.

3.2 Both behaviors in interaction

We pointed out that the operational behavior guides the performance of the control behavior of a Web service. This guidance requires bringing both behaviors together. For instance, *done* state that a Web service takes on in the operational behavior will in return make this

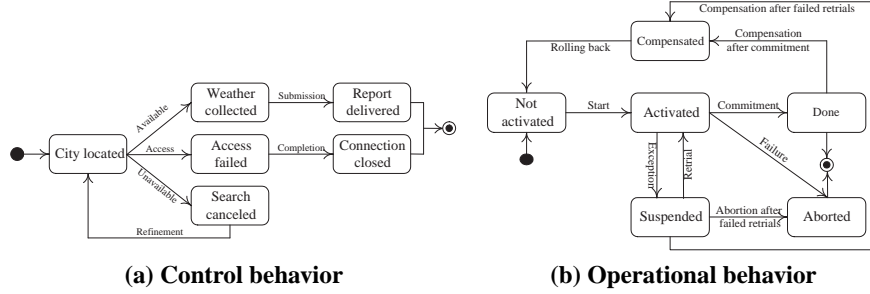


Figure 2. WeatherWS's control and operational behaviors

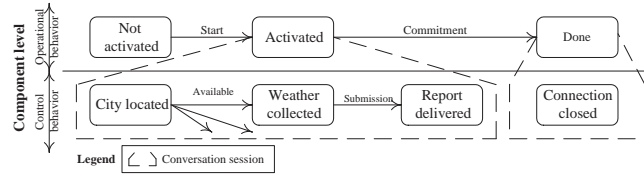


Figure 3. Example of operational and control behaviors mapping in WeatherWS

Web service take on other appropriate counterpart states like weather-collected and report-delivered in the control behavior.

The process of connecting operational and control behaviors together results in establishing *conversation sessions* between the respective states of these two behaviors (Fig. 3). To complete this connection process, a mapping function is defined as follows:

Definition 3 (Mapping Function). Let \mathcal{P}_{co} be the set of all paths in the control behavior of a Web service starting from any state in this behavior. Connecting the operational behavior to the control behavior and *vice-versa* occurs using the following mapping function: $Map : \mathcal{S}_{op} \rightarrow 2^{\mathcal{P}_{co}}$, where $2^{\mathcal{P}_{co}}$ is the power set of \mathcal{P}_{co} . \square

What the he mapping function Map does is to associate each state in the operational behavior with a set (possibly empty) of possible paths in the control behavior.

Example 4: Fig. 3 is an example of the use of the mapping function in WeatherWS where *activated* state in the operational behavior is associated with multiple paths in the control behavior. One of these paths is: $city-located \xrightarrow{l^1} weather-collected \xrightarrow{l^2} report-delivered$ where $l^1 = available$ and $l^2 = submission$. A second path for *activated* state is given in Fig. 4 (b) as well.

On top of the mapping function Map , interactions

between control and operational behaviors require a specification operation that indicates which state in the operational behavior is associated with which set of possible paths in the control behavior along with the “new” transitions that will implement these interactions. The next state to take on in the operational behavior is determined by the executed path in the control behavior and whether this execution was a success or failure. In other words, the specification operation lets the control behavior indicate to the operational behavior what needs to be done next. We define the specification operation as follows:

Definition 4 (Specification Operation). Let \mathcal{L}_S be the set of labels associated with the “new” transitions between operational and control behaviors. The specification operation uses the following two functions:

$$Spec : \mathcal{S}_{op} \rightarrow 2^{\mathcal{L}_S \times \mathcal{P}_{co} \times \mathcal{L}_S} \text{ and } Next : \mathcal{S}_{op} \times \mathcal{P}_{co} \rightarrow \mathcal{L}_{op} \times \mathcal{S}_{op}. \square$$

The specification function $Spec$ associates each state s_{op} in the operational behavior with a (possibly empty) set of triples. A triple contains (i) the label of the transition from s_{op} to the first state in the control behavior of a mapped path, (ii) the mapped path itself $p^{i \rightarrow j}$, and (iii) the label of the transition from the last state in the control behavior of the mapped path back to s_{op} in the operational behavior. We qualify the “new” transitions that connect states in independent state charts as *inter-behavior* (note that intra-behavior transition was used in Definition 1). The partial function $Next$ associates both a given state in the opera-

tional behavior and the mapped path in the control behavior with both the next state to take on in the operational behavior and the associated transition label.

Example 5: Fig. 4 shows the synchronization of WeatherWS's operational and control behaviors where two types of transitions exist: intra-behavior from $\mathcal{T}_{co} \cup \mathcal{T}_{op}$ (plain lines) and inter-behavior (dashed lines, Labels_{1,2,3}). Fig. 4 contains $Spec(activated)=\{(label_1, path_1, label_2), (label_1, path_2, label_3)\}$, $Next(activated, path_1)=(commitment, done)$,

where $path_1 = city-located \xrightarrow{l^1} weather-collected \xrightarrow{l^2} report-delivered$ and $Next(activated, path_2)=(failure, aborted)$ where $path_2 = city-located \xrightarrow{l^3} access-failed \xrightarrow{l^4} connection-closed$.

In Fig. 4, the initiation of WeatherWS is shown in the operational behavior with activated state. WeatherWS takes on this state following receipt of a user's request. Because of (activated, label₁, city-located) inter-behavior transitions, the execution of WeatherWS begins by using a dedicated database to search for the requested city. This makes WeatherWS take on city-located state in the control behavior. Afterwards, two cases are identified.

Case a. Everything goes fine and a 5-day weather-forecast report is delivered back to the user. Because of (report-delivered, label₂, activated) inter-behavior transition, this makes WeatherWS complete its operation with success by transiting from activated to done states in the operational behavior, i.e., (activated, commitment, done) intra-behavior transition.

Case b. The access to the database fails (not like in case a) as the control behavior of WeatherWS indicates with access-failed and connection-closed states. Because of (connection-closed, label₃, activated) inter-behavior transition, this makes WeatherWS terminate its operation with failure by transiting from activated to aborted states in the operational behavior, i.e., (activated, failure, aborted) intra-behavior transition.

To wrap-up this section, the formal definitions of inter-behavior and conversation session are provided. Needless to propose a formal definition for intra-behavior transition, which is a regular transition in a state chart (Definition 1).

Definition 5 (Inter-Behavior Transition). The set of all inter-behavior transitions that connect the operational and control behaviors of a Web service is denoted by \mathcal{IT} where $\mathcal{IT} = \mathcal{IT}_{op \rightarrow co} \cup \mathcal{IT}_{co \rightarrow op}$ such that: $\mathcal{IT}_{op \rightarrow co} \subseteq \mathcal{S}_{IT(op)} \times \mathcal{L}_{op \rightarrow co} \times \mathcal{S}_{IT(co)}$ is the inter-behavior transition relation starting from the operational behavior and ending at the control behavior; $\mathcal{IT}_{co \rightarrow op} \subseteq \mathcal{S}_{IT(co)} \times \mathcal{L}_{co \rightarrow op} \times$

$\mathcal{S}_{IT(op)}$ is the inter-behavior transition relation starting from the control behavior and ending at the operational behavior; $\mathcal{S}_{IT(op)} \subseteq \mathcal{S}_{op}$ is a finite set of state names in the operational behavior that take part in inter-behavior transitions; $\mathcal{S}_{IT(co)} \subseteq \mathcal{S}_{co}$ is a finite set of state names in the control behavior that take part in inter-behavior transitions; and $\mathcal{L}_{op \rightarrow co}$ is a set of inter-transitions' labels from the operational to the control behaviors, and $\mathcal{L}_{co \rightarrow op}$ is a set of inter-transitions' labels from the control to the operational behaviors ($\mathcal{L}_{co \rightarrow op} \cup \mathcal{L}_{op \rightarrow co} = \mathcal{L}_S$ (Definition 4)). \square

Before we define Web service conversation session, we introduce another function known as *Lab*. This function returns the label of an inter-behavior transition: $Lab : \mathcal{IT} \rightarrow \mathcal{L}_S$.

Definition 6 (Web Service Conversation Session). A conversation session between the operational and control behaviors of a Web service is a 4-tuple $\langle s_{op}, it_{op \rightarrow co}, p_{co}, it_{co \rightarrow op} \rangle$ such that: $s_{op} \in \mathcal{S}_{op}$, $it_{op \rightarrow co} \in \mathcal{IT}_{op \rightarrow co}$, $it_{co \rightarrow op} \in \mathcal{IT}_{co \rightarrow op}$, $p_{co} \in \mathcal{P}_{co}$; and $(Lab(it_{op \rightarrow co}), p_{co}, Lab(it_{co \rightarrow op})) \in Spec(s_{op})$. \square

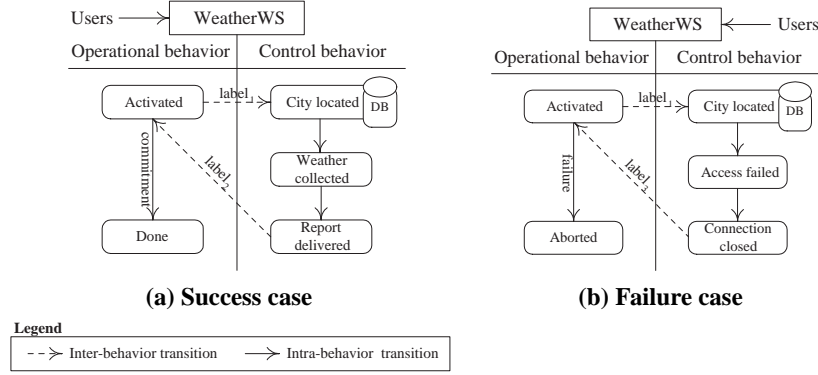
4 Synchronization woven into composition

Synchronization is a mechanism by which independent entities coordinate their next actions by agreeing on how, where, and when to carry out these actions. In the rest of this paper, entities correspond to Web services that could be either component or composite. We look into synchronization from two perspectives: *Intra*, which means how the operational and control behaviors in a component/composite Web service are coordinated (Section 4.1), and *Inter*, which means how the operational and control behaviors in separate component Web services are coordinated within the context of the same composite Web service (Section 4.2). Because composition could have either a centralized or a P2P orchestration schema, inter Web-services synchronization is examined from these two types.

4.1 Intra Web-services synchronization

Case of component Web services. The synchronization of component Web services was the main object of our research project in [9], so further details are provided in this reference. Table 1 contains some synchronization messages we developed in order to allow the operational and control behaviors interact with each other.

Case of composite Web services. The synchronization of composite Web services is differently handled from the synchronization of component Web services. This is due to the characteristics of composite Web services that need now to be highlighted through their operational and


Figure 4. Synchronization of *WeatherWS*'s control and operational behaviors
Table 1. Messages during intra (component) Web-service synchronization

#	Message name	Description
1.	<i>sync</i>	Originates from an operational state and targets a control state. The purpose is to trigger the execution of the control states (including the targeted control state) in a conversation session. <i>sync</i> is a blocking message, which makes the operational state wait for a notification back from the last control state to execute in this conversation session.
2.	<i>success</i>	Originates from a control state and targets the operational state that submitted <i>sync</i> . The purpose is to inform this operational state of the successful execution of the control states in a conversation session and to return the execution thread back to this operational state as well. <i>success</i> is coupled with <i>sync</i> .

control behaviors. These characteristics are as follows. Firstly, the control behavior represents the business logic of a composition scenario and no longer the business logic of a certain component Web service. Secondly, the current definition of the operational behavior (Definition 1) does not tell much about the execution outcome of a composition scenario and if this execution either succeeded or failed. This current definition through states like *activated* and *suspended* is geared towards the needs of the component level, only (Fig. 2 (b)).

Definition 7 (*Composite Web Service Control Behavior*). The control behavior of a composite Web service is a 5-tuple $B_{co}^{cws} = \langle \mathcal{WS}_{co}, \mathcal{L}_{co}, \mathcal{T}_{co}, \mathcal{WS}_{co}^0, \mathcal{F}_{co} \rangle$ where \mathcal{WS}_{co} is a finite set of states that correspond to Web services' names; $\mathcal{WS}_{co}^0 \subset \mathcal{WS}_{co}$ is the set of initial states that correspond to initial Web services; $\mathcal{F}_{co} \subseteq \mathcal{WS}_{co}$ is the set of final states that correspond to final Web services; \mathcal{L}_{co} is a set of labels; and $\mathcal{T}_{co} \subseteq \mathcal{WS}_{co} \times \mathcal{L}_{co} \times \mathcal{WS}_{co}$ is the transition relation. Each transition $t = (ws^{src}, l, ws^{tgt})$ consists of a source Web service $ws^{src} \in \mathcal{WS}_{co}$, a target Web service $ws^{tgt} \in \mathcal{WS}_{co}$, and a transition label $l \in \mathcal{L}_{co}$. \square

Example 6: Fig. 1 is a state chart of the control behavior of the *CookoutParty* composite Web service. Several states like *WeatherWS* (initial state) and *CateringWS* (final state) and several transitions like

(*WeatherWS*, *NiceWeather*, *PlaceBookingWS*) are included. In this transition example, *WeatherWS* and *PlaceBookingWS* are the source and target states, respectively, and *NiceWeather* is the transition's label.

Definition 8 (*Composite Web Service Operational Behavior*). The operational behavior of a composite Web service is defined as an instance of the behavior of a Web service (Definition 1) and is denoted by $B_{op}^{cws} = \langle \mathcal{S}_{op}, \mathcal{L}_{op}, \mathcal{T}_{op}, s_{op}^0, \mathcal{F}_{op} \rangle$. \square

The purpose of the operational behavior of a composite Web service is (i) to initiate the execution of its specification, which is in fact the control behavior of this composite Web service and (ii) to report on the success or failure of the execution of this specification. As a result, the operational behavior of a composite Web service is a subset of the operational behavior of a component Web service.

Example 7: Fig. 5 is another state chart that illustrates this time the operational behavior of the *CookoutParty* composite Web service. In this state chart, the number of states is limited to four, namely *not-activated*, *activated*, *done*, and *aborted*, and the number of transitions is limited to three, namely *start*, *commitment*, and *failure*.

Compared to the six states in the operational behavior of a component Web service (Fig. 2 (b)), the four states in the



Figure 5. Operational behavior of a composite Web service

operational behavior of a composite Web service (Fig. 5) puts some restrictions on the authorized synchronization messages (like those suggested in Table 1) that can be considered between this operational behavior and its counterpart control behavior. These restrictions are hereafter listed:

1. There is one conversation session between the operational and control behaviors. This session includes the *activated* state in the operational behavior and all the states (i.e., \mathcal{WS}_{co}) in the control behavior.
2. Interaction message of type *sync* to come out of the *activated* state in the operational behavior has one recipient, which is the initial state(s) (i.e., Web service) in the control behavior (i.e., \mathcal{WS}_{co}^0).
3. Any state (i.e., component Web service) in the control behavior can only submit an interaction message of type *fail* back to the *activated* state in the operational behavior. This restriction is waived for the final state(s) in the control behavior (i.e., \mathcal{F}_{co}) that can submit on top of *fail* message another message of type *success* back to the *activated* state in the operational behavior.
4. Interaction message of type *syncreq* is not allowed from the control to the operational behaviors.

4.2 Inter Web-services synchronization

Centralized orchestration is well “embraced” in Web services composition projects. But, a few projects look into the changes that need to be made in Web services standards/specifications like BPEL to smooth the design and development of P2P orchestration. Gowri Nanda et al. note that because performance and throughput are major concerns in enterprise applications, removing the inefficiencies that a centralized control introduces, is required [4]. A BPEL program could be partitioned into independent sub-programs that interact with each other without any centralized control. Gowri Nanda et al. propose a technique to partition a composite Web service written as a single BPEL program into an equivalent set of decentralized processes. This technique minimizes communication costs and maximizes the throughput of multiple instances of the input program.

In this paper, we look at inter Web-services synchronization from two perspectives: centralized and **P2P** (focus of this paper). This synchronization aims at initiating the development of composition scenarios and overseeing the execution progress of this development at run-time. As a result, this raises the necessity of enhancing Web services with additional mechanisms based on the needs and requirements of these composition scenarios. For instance, a Web service has now to decide if it would or not take part in a composition scenario subject to carrying out some sort of self-assessment [6]. That was not the case in the intra Web-services synchronization (Section 4.1) where the focus was on how to specify the execution of “isolated” Web services.

We identify the additional mechanisms that should embody Web services along four cases, which we denote by *invitation*, *execution*, *verification*, and *replacement*. These four cases abstract the different types of actions that component and composite Web services carry out during vertical and horizontal interactions. For example, a Web service should submit its performance details to a composite Web service as part of the verification exercise that this composite Web service carries out. In addition, a Web service should not leave its ongoing operations pending in case a composite Web service decides to substitute it as part of the replacement exercise. These additional mechanisms need to be woven into the business logic that underpins the functionality of a Web service. In [7], we elaborate on how this weaving should place in compliance with some design principles like separation of concern and aspect-oriented programming. To keep the paper self-contained on synchronization, enriching Web services with additional mechanisms is excluded.

Case of P2P orchestration. The synchronization of inter Web-services in a P2P orchestration reinforces the existence of the component level, only. Each component Web service that takes part in a composition scenario is associated with an operational and a control behaviors (Fig. 6). The previously proposed definitions for these two behaviors continue to be used (Definition 1). However, new definitions are deemed appropriate for first, the inter-behavior transitions between component Web services and second, the conversation sessions that result out of setting-up these inter-behavior transitions. These new definitions have to be inline with the authorized actions to carry out in a P2P orchestration. These actions are “invite”, “trigger”, and “ping”.

In Fig. 6, the double-angled lines (plain and dashed) illustrate where the synchronization of inter Web-services should take place in a P2P orchestration. Numbers associated with these lines represent message chronology. In the P2P orchestration we hereafter adopt, sequential execution of the component Web services is assumed even though concurrent execution could be handled without any substan-

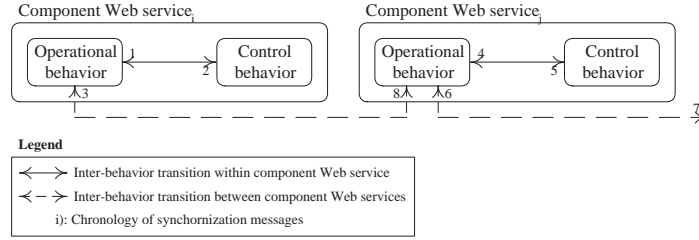


Figure 6. Synchronization of behaviors at the composition level – P2P orchestration

tial changes in the new or existing definitions. Plain lines in Fig. 6 represent inter-behavior transitions within the same component Web service (Section 4.1). What is now needed, which is the focus of this part of the paper, is to define the inter-behavior transitions between component Web services. These inter-behavior transitions are represented with dashed lines (3, 6, 7, 8) in Fig. 6.

In a P2P orchestration, the absence of a centralized coordination that would “spread the word” to other component Web services about the execution outcomes of their peers requires some changes in the way these component Web services should behave. For instance, component Web services cannot announce their immediate successful execution until they receive positive feedbacks from their peers in a reverse order. In case of negative feedbacks, these component Web services have to cancel or compensate their execution outcomes and notify their predecessors about the cancellation or compensation actions they have taken, as well. Announcement delay and backward notification have to be reflected on the operational levels of the different component Web services. Like in a centralized orchestration we split done state into two states (Fig. 7): *partial done* and *final done*.

- *Partial done* in a component Web service allows to pass on the execution thread to the next peer(s) (Fig. 6, dashed lines 3 and 6).
- *Final done* in a component Web service permits to confirm its successful execution (final completion) following receipt of a positive notification from a successor peer (Fig. 6, dashed lines 7 and 8).

Definition 9 (*Completion Status of Component Web Service in Peer-to-Peer Orchestration*). Let assume a composition scenario of n component Web services. The completion status of a component Web service WS_i in term of either success or failure is dependent on the notification message that WS_i receives from its direct successor component Web service WS_{i+1} .

$$Status(WS_i) = \begin{cases} Notify(WS_{i+1}) & i = 1, \dots, n-1 \\ success | failure & i = n \end{cases}$$

It is worth to mention that *success* and *failure* are conversational messages between component Web services. These messages are specified in Table 2.

Fig. 7 illustrates the different conversation sessions that need to be set-up in a P2P orchestration. The focus is on conversation session #2; conversation session #1 is already discussed in Section 4.1. The identification of the inter-behavior transitions that should be included in conversation scenario #2 takes advantage of the set of acceptable actions (e.g., “invite” and “ping”) that can be carried out between component Web services. These actions are now woven into the synchronization messages to occur between the respective operational behaviors of the component Web services. The following comments are made on the new operational behavior of a component Web service: (i) *partial done* and *final done* states are added and connected, (ii) *partial done* and *aborted* states are connected, and (iii) *partial done* and *compensated* states are connected as well.

Table 2 summarizes some messages that can be exchanged in a P2P orchestration during inter Web-services synchronization. This table is built upon the messages of Table 1. The description of each message type shows (i) the direction of the bidirectional flow between the operational behaviors of the component Web services, and (ii) the case that corresponds to the actions to perform during horizontal interactions. Interesting to discuss messages #9 and #10, i.e., *confirm* and *cancel*, respectively in Table 2. Both messages are used by component Web services to notify other component Web services that they could either confirm or cancel their execution.

Definition 10 (*Inter-Behavior Transition in Peer-to-Peer Orchestration*). The set of all inter-behavior transitions that connect the operational behaviors of component Web services together in a P2P-orchestration mode (conversation session #2 in Fig. 7) is denoted by $\mathcal{IT}^{(ws_i, ws_j)}$ where $\mathcal{IT}^{(ws_i, ws_j)} = \mathcal{IT}_{op \rightarrow op}^{(ws_i, ws_j)} \cup \mathcal{IT}_{op \rightarrow op}^{(ws_j, ws_i)}$ such that:

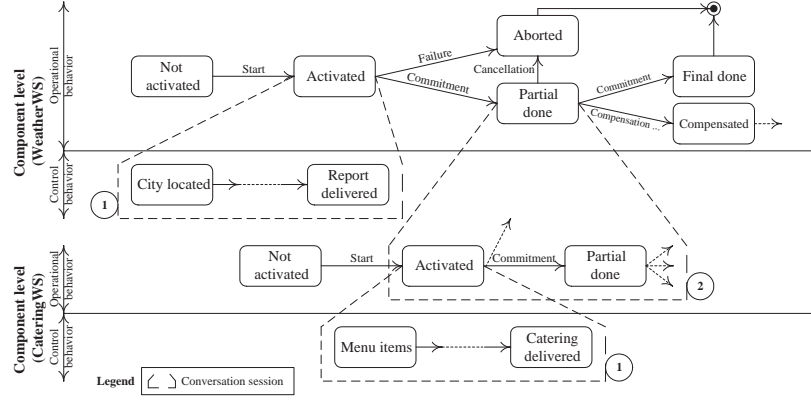


Figure 7. Operational and control behaviors mapping in *CookoutParty* - P2P orchestration

$\mathcal{IT}_{op \rightarrow op}^{(ws_i, ws_j)} \subseteq \mathcal{S}_{IT(op)} \times \mathcal{L}_{op \rightarrow op}^{(ws_i, ws_j)} \times \mathcal{S}_{IT(op)}$ is the inter-behavior transition relation starting from the operational behavior of a component Web service (WS_i) and ending at the operational behavior of another component Web service (WS_j). Same definition applies to $\mathcal{IT}_{op \rightarrow op}^{(ws_j, ws_i)} \subseteq \mathcal{S}_{IT(op)} \times \mathcal{L}_{op \rightarrow op}^{(ws_j, ws_i)} \times \mathcal{S}_{IT(op)}$; $\mathcal{S}_{IT(op)} \subseteq \mathcal{S}_{op}$ is a finite set of state names in the operational behavior of a component Web service that take part in inter-behavior transitions; and $\mathcal{L}_{op \rightarrow op}^{(ws_i, ws_j)}$ is a set of inter-transitions' labels from the operational behavior of a component Web service (WS_i) to the operational behavior of another component Web service (WS_j), and $\mathcal{L}_{op \rightarrow op}^{(ws_j, ws_i)}$ is the opposite ($\mathcal{L}_{op \rightarrow op}^{(ws_i, ws_j)} \cup \mathcal{L}_{op \rightarrow op}^{(ws_j, ws_i)} = \mathcal{L}_S$). \square

5 Implementation

To test the viability of the proposed approach, a prototype system was implemented in Java and integrated under Eclipse 3.3 by extending the Web service development platform we developed previously [9]. The prototype consists of four modules:

ControlBehaviorModeler,
OperationalBehaviorModeler,
ConversationModeler and
SimulationController.

The ControlBehaviorModeler and the OperationalBehaviorModeler assist engineers specify the control and operational behaviors of a component or a composite Web service, respectively. In particular, we developed a visual interface for editing Web services' behaviors using state charts. The ConversationModeler takes the behavior specifications of a Web service as an input to produce conversation specifications (i.e., inter-transitions and message se-

quences). It implements functions to support conversations between operational and control behaviors. Specifically, it provides methods for managing conversation instances and triggering transitions. When dealing with a peer to peer synchronization, the ConversationModeler manages first, the inter-behavior transitions between component Web services and second, the conversation sessions that result out of setting-up these inter-behavior transitions. Finally, the SimulationController tracks and analyzes (if necessary) the execution of a composite or a component Web service according to its conversation definition (e.g., whether the messages are received and sent in an appropriate order).

Fig. 8 shows the execution of a component Web service in the case of a peer to peer orchestration. Upon the reception of a user's request, the operational level of the first component Web service (*WeatherWS* in this case) moves from not-activated state to activated state (red color is used to show the execution path). This latter state, submits a Sync message to Citylocated state in the control behavior to trigger its execution. In a success case, Reportdelivered state returns a Success message back to the activated state in the operational behavior. Based on this information, the operational behavior moves from activated state to partial done state. This latter state sends a trigger message to invoke the next component Web service (*CateringWS*).

6 Conclusion

We presented in this paper a framework for establishing synchronization between Web services engaged in composition scenarios. Synchronization assists independent parties coordinate their actions and thus, avoid conflicts. This framework extends the research work we carried out on iso-

Table 2. Messages during inter Web-services synchronization - P2P orchestration

#	Message name	Description			
		From	To	In reply to	Case
1.	invite	Component $WS_i (\mathcal{B}_{op})$	Component $WS_j (\mathcal{B}_{op})$	null	Invitation
2.	trigger	Component $WS_i (\mathcal{B}_{op})$	Component $WS_j (\mathcal{B}_{op})$	null	Execution
3.	ping	Component $WS_i (\mathcal{B}_{op})$	Component $WS_j (\mathcal{B}_{op})$	null	Verification
4.
9.	confirm	Component $WS_i (\mathcal{B}_{op}^{WS})$	$1[\text{Component } WS_j (\mathcal{B}_{op})](i-1)$	success	Execution
10.	cancel	Component $WS_i (\mathcal{B}_{op}^{WS})$	$1[\text{Component } WS_j (\mathcal{B}_{op})](i-1)$	failure	Execution

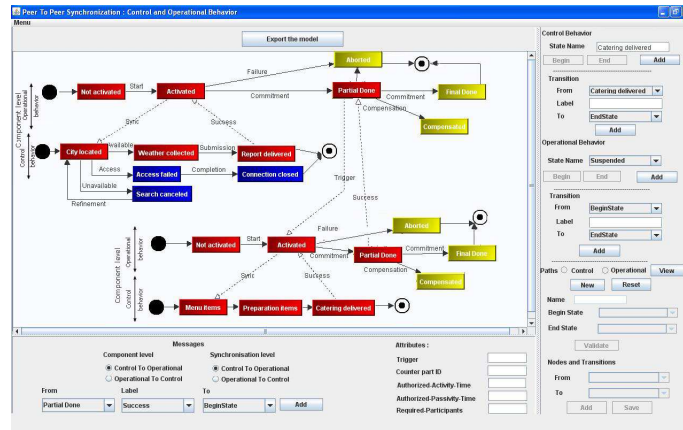


Figure 8. An execution of a component Web service in a P2P orchestration

lated Web services (not engaged in any composition) and leverages two types of behaviors related to specifying such Web services. The control behavior defines the business logic that underpins the functioning of a Web service, and the operational behavior regulates the execution progress of this control behavior by stating the actions to carry out and the constraints to put on this progress. Synchronizing Web services through their respective behaviors has revealed that the orchestration schemas, whether centralized or P2P, affect the mechanisms to develop in response to the needs and requirements of the composition scenario that is under consideration. The use of some of these mechanisms per orchestration schema was demonstrated through a prototype. In term of future work, we plan to study the value-add of model checking to the early-detection of design inconsistencies and errors.

References

- [1] Benatallah, B., Q. Z. Sheng, Ngu A. H. H., and M. Dumas. Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'2002)*, San Jose, CA, US, 2002.
- [2] F. Casati and M. C. Shan. Dynamic and Adaptive Composition of E-Services. *Information Systems*, 26(3), 2001.
- [3] F. Daniel and B. Pernici. Insights into Web Service Orchestration and Choreography. *International Journal of E-Business Research, The Idea Group Inc.*, 1(2), 2005.
- [4] M. Gowri Nanda, S. Chandra, and V. Sarkar. Decentralizing Execution of Composite Web Services. In *Proceedings of 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'2004)*, Vancouver, British Columbia, Canada, 2004.
- [5] M. Kavas, J. Bentahar, Z. Maamar, and H. Yahyaoui. Formal Verification of Conversations in Composite Web Services using NuSMV. In *Proceedings of the 8th International Conference on Software Methodologies, Tools and Techniques (SoMeT09)*, IOS Press, Prague, Czech Republic, 2009.
- [6] Z. Maamar, S. Kouadri Mostéfaoui, and H. Yahyaoui. Towards an Agent-based and Context-oriented Approach for Web Services Composition. *IEEE Transactions on Knowledge and Data Engineering*, 17(5), May 2005.
- [7] Z. Maamar, M. Sheng, D. Benslimane, and H. Yahyaoui. Web Services Interactions: Analysis, Modeling, and Management. *International Journal on Software Engineering and Knowledge Engineering, World Scientific Publishing Co.*, 18(2), March 2008.

- [8] Z. Maamar, M. Sheng, H. Yahyaoui, J. Bentahar, and K. Boukadi. A New Approach to Model Web Services' Behaviors based on Synchronization. In *Proceedings of the 23rd IEEE International Conference on Advanced Information Networking and Applications, Symposium on Frontiers of Information Systems and Network Applications*, Bradford, UK, 2009.
- [9] Z. Maamar, M. Sheng, H. Yahyaoui, J. Bentahar, and K. Boukadi. Conversation-Oriented Engineering of Web Services. Technical report, College of Information Technology, Zayed University, September 2007.
- [10] Q. Z. Sheng, B. Benatallah, Z. Maamar, M. Dumas, and A. H. H. Ngu. Enabling Personalized Composition and Adaptive Provisioning of Web Services. In *Proceedings of the 16th International Conference on Advanced Information Systems (CAiSE'2004)*, Riga, Latvia, 2004.
- [11] W. Yang and S. Tang. A Solution for Web Services Transaction. In *Proceedings of The 2006 International Conference on Hybrid Information Technology (ICHIT'2006)*, Cheju Island, Korea, 2006.

Enhancing Engineering Methodology for Communities of Web Services

M. El-Menshawy

Depart. of Electrical and Computer Engineering
Concordia University, Montreal, Canada
m_elme@encs.concordia.ca

J. Bentahar, R. Dssouli

Concordia Institute for Inf. Sys. Engineering
Concordia University, Montreal, Canada
{bentahar, dssouli}@ciise.concordia.ca

Abstract—Communities of web services have been proposed to gather web services having the same functionalities but possibly different nonfunctional properties. Current approaches into communities of web services focus on developing, managing and designing communities of web services through a suitable architecture, but can benefit from a stronger treatment of flexible interactions. These approaches ignore the collaboration and business-level contracts between various web services and the ability to formally delegate service to another web service within the same community. This paper presents a significant step towards enhancing communities of web services using an agent-based approach that synthesizes mentalistic states (e.g. goals, tasks), social commitments and argumentative dialogues for modeling and establishing communities of web services. This paper has three contributions: first, we extend the community structure with alliances structure to allow collaboration between various web services; second, we propose a new engineering methodology based on concepts of Tropos methodology for managing communities of web services with alliances structure; and third, we specify internal-organizational business interactions within web services in terms of commitments augmented with argumentative dialogues to reason about the validity of these commitments. We evaluate our methodology using a large existing case study of auto insurance claim processing.

I. INTRODUCTION

The notion of community has been proposed to gather web services having the same functionalities independently of their origins and the way they carry out these functionalities [3], [4], [11]. In recent years, the capability of argumentation and dialogue games has been used in managing and reconciling conflicts of interests that may arise within a community of web services. In this context, the argumentation theory allows agent-based web services to interact rationally, argue about the reasons that support or disavow their conclusions, persuade a new web service to join a community, negotiate with other peers to reach a deal and assail each other through an attack-binary relation as well as specify the interaction mechanisms within communities [3], [4].

In fact, web services of business scenarios begin with a user's need and end with a user's need fulfillment. The structure of community facilitates and speeds up the process of web services discovery in open settings and helps in selecting the best ones for composite business scenarios [12], [14] when users' requests cannot be satisfied by a single available web service but need collaboration among available web services to handle them [11].

Recently, a certain number of significant proposals have been introduced [1], [3], [4], [10]–[13] to address the modeling and management issues of communities of web services (CWSs) in order to allow them to interact more flexibly for the growing needs of business processes. However, the main objective of modeling and managing CWSs and the collaboration between web services in an efficient way along with business relationships has not been reached yet. In particular, we have three broad elements which should be addressed: designing, engineering, and managing communities.

- The architecture based on traditional software engineering methodologies proposed in [3], [4], [11] lacks business-level contracts that represent underlying interactions between web services. This architecture depends on message occurrence and ordering irrespective of the message meanings, thereby a message-based approach hides many details of the internal organization of the architecture that affect designing business processes.
- Existing methodology [4], [11] for modeling, operationalizing, and evolving CWSs ignores formalizing the delegation process that is used to delegate incomplete services within community, in the case of replacing misbehaving web services and keeping the system reliable. Furthermore, this methodology does not consider the collaboration among members of the community and the mentalistic states of agent-based web services. In general, the approaches discussed in the literature consider only two-party operations, while real-life scenarios are typically multiparty operations that need a mediator agent-based web service to complete users' requests.
- The process of handling failures and exceptions is very hard to implement within the current structure of the community. Moreover, the number of interactions between members of the community needs to be reduced to enhance the response time of participating web services.

This paper aims to enhance the community structure by setting up alliances structure among web services to overcome the aforementioned shortcomings in [3], [4], [11]. Thereby the resulting community structure becomes more realistic with the real-life business scenarios in distributed systems. By so doing, we propose a new engineering methodology based on Tropos methodology [17], which we enhance with concepts

of communities, commitments and argumentative dialogues for modeling CWSs extended with alliances structure. In fact, this work is an extension of our previous research in which we have modeled and specified CWSs based on argumentation capabilities. This model enables web services through associated agents to argue, persuade and negotiate with their peers using a dialectical process to satisfy their goals in an efficient way [3], [4]. Specifically, here we reconfigure the design of community with alliances structure from the perspective of the collaboration between agent-based web services that participate in composition business scenarios and benefit from agent reasoning capabilities about nonfunctional properties.

The contributions of this paper are manifold: (i) alliances structure within CWSs enhances response time of web services that handle users' requests and reduces the number of interconnections between members of community; (ii) a new agent-based engineering methodology synthesizing mentalistic states, social commitments and argumentative dialogues; and (iii) a composition mechanism allowing agent-based web services to collaborate with each other in the form of delegation operations. Additionally, we present a case study of auto insurance industry scenario to evaluate our methodology.

The remainder of this paper is organized as follows. Section II introduces the notion of alliance structure and key concepts for our methodology. Section III presents our engineering methodology for developing communities of web services. Our case study is presented in Section IV to evaluate different steps of our methodology. The paper ends in Section V with the relevant literature discussions and future work directions.

II. ALLIANCE STRUCTURE AND CONCEPTS

This section defines alliances structure within communities of web services and key concepts to be used clearly in our methodology.

A. Alliance Structure

In [3], [4], [11] the notion of communities of agent-based web services has been introduced. Here we would prescribe how to extend it via introducing alliances structure. In essence, the service providers over forced to improve their strategies and redistribute business functionalities to be able to compete with others should think about building alliances. An alliance structure based on Quality of Service (QoS) is the concept that reconfigures community structure to achieve competitive pressures between service providers and collaboration between agent-based web services. In [3], [11] an alliance structure as a subset of community or a micro-community based on mutual agreements between providers of web services as part of their partnership strategies has been superficially introduced.

We develop this view by considering nonfunctional properties (e.g., QoS, reputation, response time) associated with each agent-based web service as a vital principle to cluster two or more web services into different alliances as the second level of community, since the first level is occupied by master web service (see Fig.1). Whereas the third level of community contains web services within alliances that underpin the same

or part of the community's functionality. These web services need to collaborate with other peers to achieve the whole or global community's functionality.

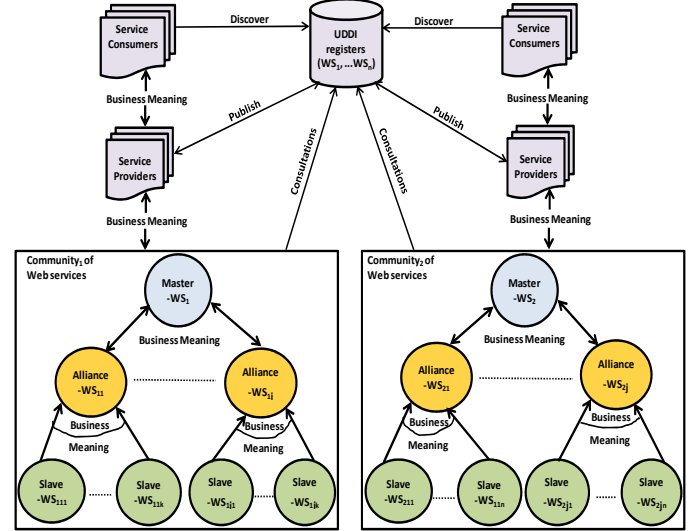


Fig. 1. An architecture of communities extended with alliances structure.

For example, in the purchase scenario of goods, the functionality of a community is purchasing goods. This community combines web services having complementary functionality within an alliance, such as *placing an order*, *paying*, and *shipping* with a high reputation and response time. Notice that, these agent-based web services have logically distinctive functionalities and involve distinctive roles from the community's functionality but they can collaborate with each other to achieve a goal by combining or composing their functionalities (i.e., purchasing goods) and to satisfy the community's functionality.

As a result the new internal organization of the community with alliances structure make it easier to detect failures and errors as we search in micro-communities and reduce the number of interactions between members of the community since the master web service has one connection with each alliance structure instead of a direct connection with each web service (i.e., this minimize the overhead). Moreover, an alliance inherits a dynamic property from its community where new members can admit to or exclude from the alliance, and alliances themselves can either discard or merge at any-time.

B. Key Concepts

The key concepts of our methodology are inspired by Tropos methodology [17] with the extra flexibility resulting from considering new concepts such as community, commitment and argumentative dialogue. The purpose is to enhance Tropos capabilities in order to deal with the intrinsic complexity of business processes.

1) **Community**: a collection of web services with similar or part of total functionality organized into three broad levels without explicitly referring to concrete web services that will implement this functionality within alliances structure at run-time (see Fig.1).

2) **Agent**: a computational representation of web service within community structure with metaphors that make it appropriate for developing, designing and implementing distributed business systems. Each agent has strategic goals and capabilities to execute tasks. It engages with others in business relationships to get other agents performing the delegated tasks on its behalf and to reason about the validity of their tasks.

3) **Role**: an abstract entity over agent-based web services and can be specified by their related sets of commitments augmented with argumentation systems within a community. Multiple agents could play one role and different roles could be played by a single agent in a community.

4) **Goal**: a state of the world that an agent would like to reach or bring about. In other words, a goal is a final or an acceptance state. Tropos methodology defines two types of goals: *hard goals* are functional requirements and often have a measurable satisfaction condition that can be satisfied. The latter one, called *soft goals*, no precise criteria for its satisfaction can be found, these goals model nonfunctional requirements of the community. The AND/OR decomposition is used to decompose a root goal into subgoals.

5) **Task**: an abstract method by which a goal can be achieved. Parallel with the concept of goal, there are two types of tasks to achieve the companion goals. Here we also use AND/OR decomposition to decompose a root task into subtasks compatible with subgoals. The AND requires all subtasks; OR requires one subtask.

6) **Dependency**: used to identify the dependent relationship between two roles where one role (the depender) depends upon the other (the dependee) in order to achieve a goal or execute a task. This relation is written as *the depender depends upon the dependee*.

7) **Commitment**: a commitment $C(id, dbtr, cdtr, C_x, \psi, \phi)$ means that the debtor *dbtr* is responsible to the creditor *cdtr* within community context C_x for satisfying the content ϕ if the condition ψ holds. The commitment has the form of contractual style where *id* is the unique identifier for commitment, *dbtr* is the debtor role, *cdtr* is the creditor role, the context C_x may be an institution, a company, organization, marketplace (e.g., eBay) in which ongoing interactions occur, and ϕ and ψ are formulas in a given formal language.

Commitments capture state of the dependencies relation between roles and allowing a variety of possible manipulation based on a set of operations. For instance, the debtor is able to *create*, *fulfill*, *violate* commitments, *withdraw* from commitments and *delegate* commitments to another agent. Whilst the creditor has the right to *release* the debtor from commitments and *assign* commitments to another agent [20].

8) **Argumentative dialogue**: a dialectical process for the exchange of various arguments for and against some conclusions [3], [4]. Indeed, argumentation provides agent-based web services with an effective means to reconcile conflicts, seek information, persuade and negotiate with other peers within the same alliance structure. It relies on actions on commitments to generate a suitable set of arguments during dialogues to achieve mutually acceptable agreements between agent-based

web services.

III. ENGINEERING METHODOLOGY FOR WEB SERVICES COMMUNITIES

Having captured the core concepts of our methodology, here we introduce the proposed methodology that intended to support all phases of developing CWSs based on the notion of Tropos methodology. The latter one has been developed in [17] as an agent-oriented software methodology in terms of goal, task and dependency. Tropos has been enhanced with commitments to capture business interactions among partners with high-level business meaning [15]. In the same direction, this paper improves the last version of Tropos introduced in [15] with arguments and argumentative dialogues to increase its capabilities via enabling agent-based web services to argue, seek information and negotiate with other peers about the compliance of their commitments, thereby increasing its practicality in distributed business systems. Table (I) summarizes the steps in our proposed methodology. The subsections below describe a step-by-step way the progress of our methodology.

A. Requirements Analysis

This phase enhances the early and late requirements in the phases of Tropos methodology with a community concept.

1) **Step 1: Identify Community**: the engineer initially concerns with understanding the organizational context of community that gathers agent-based web services. This step includes substeps to be completed.

1.1) *Define the functionality* (e.g., *hotels booking*, *weather forecasting*, etc.) of community by binding to a specific ontology [13] (e.g., Web Ontology Language (WOL)). This binding is important since providers of web services use different terminologies to describe the functional and nonfunctional requirements of their respective web services.

1.2) *Identify agents and roles* in a community using terms like *master*, *alliance* and *slave* web service. The master web service plays the main role in a community and refers to a special web service that leads the community as well as it takes over multiple responsibilities (e.g., checking the credentials of alliance web services before they are established in the community). The other of web services in the community are slave web services.

1.3) *Identify alliance web services* by clustering two or more slave web services having the same nonfunctional properties satisfy the users' requests and having different functionality. For example, the master web service gathers slave web services that have a high QoS in the first group under the management of *alliance web service1* and the other slave web services that have a medium QoS in the second group under the management of *alliance web service2*, etc. (see Fig.1).

1.4) *Specifying dismantle community*: the master web service is only responsible for dismantling community when all alliance structures present low precision to users' requests (i.e., irrelevant results) and low recall (i.e., missing relevant information). This happens when the number of slave web services within alliances structure is not enough to satisfy users' requests as well.

2) **Step 2: Determine Goals and Dependencies:** this step iteratively determines the goal dependencies between the roles. First, it searches UDDI registries to find posted services based on the similarity that exists between users' requests and these registered services. Thus, the composite goal of users' requests is identified at the master web service as a hard goal. Second, the master web service uses nonfunctional properties to refine its populated alliance web services into one that satisfies this composite goal (say G). Third, using means-end analysis to decompose this composite goal into subgoals (say $G = \{g_1, g_2, \dots\}$), thereafter, alliance web service introduces roles that adopt these subgoals. This iterative analysis continues decomposing these subgoals until no new goal dependencies arise.

Description	Input	Output
1.1) Identify community function	A specific ontology and composition scenario.	All agent-based web services having the same or part of community's functionality.
1.2) Identify agents and roles in community	Agent-based web services and composition scenario.	Master web service and slave web services.
1.3) Identify alliance web services	Slave web services, composition scenario and nonfunctional properties.	Alliance web services and their slave web services.
1.4) Specify dismantling community	Request from alliance web service to its master web service.	Terminating community.
2) Identify goals and goal dependencies	Alliance web services, slave web services, composition scenario and goals are introduced in architecture.	Goals and goal dependencies of each dependee role (in step 1.2).
3) Identify tasks and task dependencies	Alliance web services, slave web services, goal dependencies and composition scenario.	Tasks and task dependencies.
4) Identify commitments	Tasks dependencies, scenario commitments description.	Commitments describing business relation.
5) Identify argumentative dialogues	Nonfunctional properties and commitments.	Accessible commitments.

TABLE I
OUTLINE THE STEPS IN OUR METHODOLOGY.

3) **Step 3: Identify Tasks and Dependencies:** each role from step (2) has goal dependencies, then the ultimate objective of this step is to find task dependencies that will be responsible to achieve goals dependencies. Meanwhile, one goal (say g_1) may need a set of tasks (say $t_1 = \{t_{1.1}, t_{1.2}, \dots\}$) to accomplish it. Subsequently, means-end analysis needs to identify this set of the tasks. Similarly, the task may be decomposed into subtasks and this decomposition analysis will iterate until no new task dependencies arise.

B. Architecture Design

The architectural design phase plays a crucial role in the design process. Initially, it defines the organization of the

system in terms of the components and their interdependencies that are identified in previous phase. This step focuses on how system components work together to constitute a multi-agent system and introduces resources, goals and roles as needed. This paper presents an architecture to tackle pitfalls of standard approaches that do not underpin business meaning and dynamic composition of existing services in which the components are agent-based web services and their interdependencies are specified in terms of commitments augmented with argumentation capabilities to reason about the validity of these commitments (we will explain commitment and argumentation later on). In fact, this architecture is a call and return style in the form of layered phases and an extension to the architecture we developed in [3], [4].

From Fig. 1 the main components of the proposed architecture are service providers, service consumers of web services, UDDI registers and communities with alliances structure. A community with alliances structure is organized dynamically according to the specifications discussed in previous phase (III-A). Hereafter we focus on operationalizing the steps through which the goal dependencies are to be fulfilled. The service providers publish and register the name of their services in UDDI registries with different nonfunctional properties (e.g., QoS) so that service consumers or users can search for appropriate QoS. More precisely, there are three kinds of agent-based web services (*master-ws*, *alliance-ws* and *slave-ws*) constitute the structure of community and collaborate with each other to achieve users' requests.

A *master-ws* can be implemented as a web service for compatibility purposes with the slave web services and alliances web services that populate the community as well. It delegates goal dependencies to *alliance-ws* that will be responsible for accomplishing them. The alliance web services manage micro-communities and decompose goal dependencies to slave web services that populate their alliance structure as well. Each *slave-ws* signs up contract with its alliances to commit to satisfy the delegated subgoal. The slave web services collaborate with each other to achieve these goals. Of course each slave web service has the ability to delegate or assign incomplete tasks to other slave web services to complete its goal. Moreover, *alliance-ws* can request from *master-ws* to search for a new *slave-ws* to join in its structure instead of the existing *slave-ws* that it does not work well. When all slave web services satisfy their contracts, their alliance web services consequently achieve their composite contracts. Hence these alliance web services need to inform the *master-ws* with results to finalize users' requests.

C. Detailed Design

This phase is intended to introduce additional details for each architectural component of a community structure. To support this phase, we adopt social commitment, argumentative dialogues and dialogue games protocol from the agent programming community.

1) **Step 4: Identify Social Commitments:** our methodology captures high-level business meaning via identifying commit-

ments between roles in terms of tasks. More precisely, this step analyzes each task dependency resulted from step (3) and that discovered in the architecture phase to identify the corresponding commitment. The notion here depends on the task dependency such that when the dependee (or the debtor) commits towards the depender (or the creditor) to execute the dependum task (e.g., t_1) or the commitment content, then a commitment exists. The condition of the commitment is defined by identifying a task that dependee needs to satisfy to perform commitment content. Meanwhile, the creditors need to verify some constraints that are conjuncted with this commitment to guide them to determine the corresponding tasks resulting from executing this commitment. Whereas, if the dependee is not committed towards the depender to execute a task, then no commitment exists, in this case the dependee executes the intrinsic task to achieve its internal goal.

2) **Step 5: Identify Argumentative Dialogues:** this step identifies arguments that can be used either in negotiation, persuasion or information seeking dialogue based on commitments identified in step (4).

Argumentative Locutions	Descriptions
Open-dialogue	A special argumentative act used to open the dialogue.
Accept($Ag-ws_2, C(id_x, \phi_x)$)	When $Ag-ws_2$ has an argument in favor of ϕ_x .
Refuse($Ag-ws_2, C(id_x, \phi_x)$)	When $Ag-ws_2$ has an argument against ϕ_x .
Attack($Ag-ws_2, C(id_x, \phi_x), C(id_y, \phi_y)$)	When $Ag-ws_2$ attacks the content of $C(id_y, \phi_y)$ by the content of its commitment $C(id_x, \phi_x)$.
Challenge($Ag-ws_2, C(id_x, \phi_x)$)	When $Ag-ws_2$ has neither an argument for ϕ_x nor for $\neg\phi_x$, then it challenges ϕ_x .
Justify($Ag-ws_1, C(id_x, \phi_x), C(id_y, \phi_y)$)	When $Ag-ws_1$ has a commitment $C(id_x, \phi_x)$ to justify another commitment $C(id_y, \phi_y)$.
Make-Offer($Ag-ws_1, C(id_x, \phi_x)$)	An $Ag-ws_1$ makes an offer ϕ_x to $Ag-ws_2$ when $Ag-ws_1$ has an argument in favor of ϕ_x .
Close-dialogue	A special argumentative act used to close the dialogue.

TABLE II
THE NATURAL DESCRIPTION OF THE ARGUMENTATIVE LOCUTIONS.

This methodology terms these arguments by *social arguments*, not only to emphasise their ability to resolve conflicts within a social community, but also to highlight the fact that two agent-based web services having task dependency can negotiate or persuade and, upon agreement, commit to each other for the specified value transfers. However, identifying such arguments is not merely the last step. Since agent-based web services need a language to express these arguments. In [4] we proposed *Horn logic language* to allow agent-based web services to express their arguments and to develop their reasoning capabilities within an argumentation system.

The formal specification of these arguments is defined in a dialogue game protocol, namely Persuasive-Negotiation Protocol for CWSs (*PNP-CWS*) [4]. We have eight argumentative locutions: $\{Open, Accept, Refuse, Make-Offer,$

$Attack, Challenge, Justify, Close\}$ forming the basic building blocks of this protocol. Here we present the description of these locutions in natural language (see Table (II)). To simplify the notation, a commitment will be denoted by $C(id_x, \phi_x)$ when the participating agents and the other elements are clear from the context. These locutions specify our communication language that can be used to communicate about satisfying goals or tasks.

Definition 1: (Dialogue game) Let $Open(Ag-ws_1, C(id_k, Ag-ws_1, Ag-ws_2, C_x, \psi, \phi))$ be the opening action performed by $Ag-ws_1$ and sent to another agent $Ag-ws_2$ about content ϕ subject to ψ within alliance context C_x . A dialogue game Dg is a conjunction of rules, where the first rule defines the condition to enter Dg if the argumentation systems of $Ag-ws_1$ and $Ag-ws_2$ support the satisfaction of the commitment condition ψ . The other rules identify possible actions that an agent-based web service can use as a reply when receiving an action from another agent-based web service if a given condition $Cond_{i,j}$ is satisfied. This conjunction is specified as follows:

$$\begin{aligned}
 \text{Entry rule : } & Open(Ag-ws_1, C(id_k, Ag-ws_1, Ag-ws_2, C_x, \psi, \phi)) \\
 & \wedge Cond_k \\
 \text{Body rules : } & \bigwedge_{0 < j \leq N} (Action_i(Ag-ws_p, Ag-ws_m, C(id_x, \phi_x), \\
 & C(id_y, \phi_y)) \wedge Cond_{i,j} \\
 & \Rightarrow Action_{ij}(Ag-ws_n, Ag-ws_o, C(id_z, \phi_z), C(id_w, \phi_w)))
 \end{aligned}$$

In the entry rule, the *Open* action represents the opening of the dialogue game and is executed just one time at the beginning of the dialogue. The $Cond_k$ has two possibilities either the commitment condition ψ is true (i.e., can be generated from argumentation systems of $Ag-ws_1$ and $Ag-ws_2$) or false. The body rules are executed many times during the dialogue game. In these rules, the $Action_i(Ag-ws_p, Ag-ws_m, C(id_x, \phi_x), C(id_y, \phi_y))$ is an action of type i on the propositional commitment, since the commitment condition holds, where $Action_i \in \{Create, Fulfill, Violate, Release, Withdraw, Delegate, Assign, Accept, Refuse, Attack, Justify, Challenge, Make-Offer\}$. Moreover, the bold elements meaning that they could be removed. For example, if $Action_i \notin \{Delegate, Assign\}$, then the element $Ag-ws_m$ could be removed. The $Action_{ij}(Ag-ws_n, Ag-ws_o, C(id_z, \phi_z), C(id_w, \phi_w))$ is an action on the commitment of type j with content ϕ_w that depends on the action of type i , where $Action_{ij} \in \{Create, Fulfill, Violate, Release, Withdraw, Delegate, Assign, Accept, Refuse, Attack, Justify, Challenge, Make-Offer\}$. We notice that $n = w$ when $C(id_x, \phi_x)$ does exists (e.g., Attack and Justify), otherwise, $w = y$. We also have $n = m$ if $Ag-ws_m$ does exist (e.g., Delegate and Assign), \Rightarrow is the implication symbol for dialogue game rules, and N is the number of allowed actions that $Ag-ws_n$ can perform after receiving an action from $Ag-ws_p$ where $n, p \in \{1, 2\}$.

The commitment condition and content in this definition are Horn formulas, $Cond_k$ defines the possibility of entering

Dg when the condition of the commitment is satisfied. While $Cond_{ij}$ is expressed in terms of the possibility of generating an argument from the argumentation system.

To clarify the relationship between commitment and argumentation system, let us first define some notions used in our approach. The knowledge base of agent-based web service ($Ag-ws$) is denoted by $KB(Ag-ws)$, it contains all information related to functional and nonfunctional requirements of $Ag-ws$. The argumentation system of $Ag-ws$ is denoted by $Arg_{sys}(Ag-ws)$ using Horn logic language where $\phi \triangleleft Arg_{sys}(Ag-ws)$ denotes the fact that a Horn propositional formula ϕ can be generated from argumentation system of $Ag-ws$. While the formula $\neg\phi \triangleleft Arg_{sys}(Ag-ws)$ indicates that ϕ cannot be generated from argumentation system of $Ag-ws$.

The following is an example of dialogue game within alliance context C_x , in which after opening the dialogue, an alliance web service1 ($alliance-ws_1$) invites a slave web service1 ($slave-ws_1$) to join a current composition scenario where $alliance-ws_1$ knows that $slave-ws_1$ has a reasonable role to complete this scenario. The invitation is modeled using *Make-Offer* locution. Then $slave-ws_1$ searches its $Arg_{sys}(slave-ws_1)$ to decide either to accept or refuse the invitation. $slave-ws_1$ accepts the invitation by verifying some constraints related to the performance of $alliance-ws_1$. That is, if the invited web service ($slave-ws_1$) has an argument favoring the received invitation (e.g., $alliance-ws_1$ has a good reputation) and does not have any argument against this invitation (e.g., $slave-ws_1$ does not commit to join any other alliance). Thereby $slave-ws_1$ has two choices either to accept this invitation, then $slave-ws_1$ creates a commitment towards $alliance-ws_1$ to join this composition scenario or to refuse, then $slave-ws_1$ releases from this commitment. The formal representation of the acceptance case of entering and accepting the invitation is as follows:

Example 1:

- 1) $Open(alliance-ws_1, C(id_1, alliance-ws_1, slave-ws_1, \psi, \phi)) \wedge (\psi \triangleleft Arg_{sys}(slave-ws_1)) \wedge (\psi \triangleleft Arg_{sys}(alliance-ws_1))$
- 2) $Make-Offer(alliance-ws_1, C(id_1, \phi))$
 $\wedge (Reputation-of-alliance = Good \triangleleft Arg_{sys}(slave-ws_1))$
 $\wedge (\neg Commit-to-Join-an-alliance \triangleleft Arg_{sys}(slave-ws_1))$
 $\Rightarrow Accept(slave-ws_1, C(id_1, \phi))$
where $\psi = functionality-matches-composite-scenario$
 $\phi = Invitation-for-joining$

The commitment manipulation underpins our approach with a simple mechanism of composite services between slave web services to handle users' requests through three-party actions (e.g., delegation, assignment). In simple case of delegation action (i.e., without metacommitment), we have two commitments among three agent-based web services where if an agent cannot able to complete its service, then it delegates the service to another agent (this delegation is made randomly). To clarify our notion about composing services, suppose $slave-ws_1$ is committed to $alliance-ws_1$

to bring about some facts within alliance context (C_x), if the condition ψ holds. Meanwhile, the argumentation systems of $slave-ws_1$ and $alliance-ws_1$ support this condition and the argumentation system of $slave-ws_1$ favors the content of this commitment. However, if any reason after creating the commitment, $slave-ws_1$ cannot complete it, then it will delegate the commitment to another agent-based web service in the same alliance (say, $slave-ws_2$). When $slave-ws_2$ observes the delegated commitment, it uses its argumentation system to search if it has an argument supporting the condition of this commitment. Also, if the argumentation system of $alliance-ws_1$ is still supporting the condition of this commitment, the $slave-ws_2$ will create a new commitment towards $alliance-ws_1$ (see Fig.2), formally:

Example 2:

- 1) $Open(alliance-ws_1, C(id_1, alliance-ws_1, slave-ws_1, C_x, \psi, \phi)) \wedge (\psi \triangleleft Arg_{sys}(alliance-ws_1)) \wedge (\psi \triangleleft Arg_{sys}(slave-ws_1))$
- 2) $Make-Offer(alliance-ws_1, C(id_1, alliance-ws_1, slave-ws_1, \phi)) \wedge (\phi \triangleleft Arg_{sys}(slave-ws_1))$
 $\Rightarrow Accept(slave-ws_1, C(id_1, slave-ws_1, alliance-ws_1, \phi))$
- 3) $Accept(slave-ws_1, C(id_1, slave-ws_1, alliance-ws_1, \phi))$
 $\wedge (\alpha \triangleleft Arg_{sys}(slave-ws_1))$
 $\Rightarrow Delegate(slave-ws_1, slave-ws_2, C(id_1, slave-ws_1, alliance-ws_1, C_x, \phi)) \wedge (\psi \triangleleft Arg_{sys}(alliance-ws_1))$
 $\wedge (\psi \triangleleft Arg_{sys}(slave-ws_2))$
- 4) $Delegate(slave-ws_1, slave-ws_2, C(id_1, slave-ws_1, alliance-ws_1, C_x, \phi)) \wedge (\phi \triangleleft Arg_{sys}(slave-ws_2))$
 $\Rightarrow Create(slave-ws_2, C(id_2, slave-ws_2, alliance-ws_1, C_x, \phi))$

where $\psi =$ the condition of the commitment

$\phi =$ bring about some facts

$\alpha = \phi$ cannot be completed by $slave-ws_1$ and

ϕ can be per formed by $slave-ws_2$

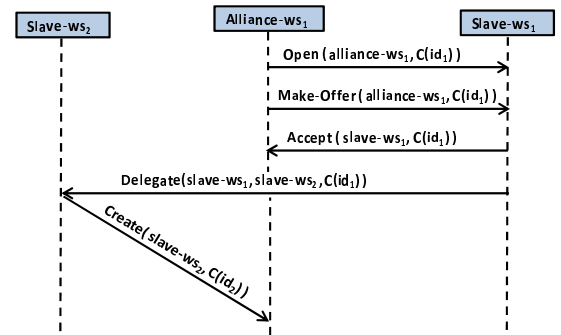


Fig. 2. The sequence diagram of the delegation action.

IV. CASE STUDY

To illustrate the application of our methodology, we consider a real-life insurance claim processing that has been studied under the *CrossFlow* project [19] and presented in many works to manage business process (see [6], [15], [16]). Fig.3 shows the use case of this case study, which is about an

insurance company in Ireland (AGFIL) including the parties involved with their individual processes. AGFIL underwrites automobile insurance policies and covers losses incurred by customers. Europ Assist (EA) provides a 24-hour help-line service for receiving customer claims and assigns name of an approved repairer to customer. Lee CS is a consulting service that coordinates with AGFIL in receiving invoices and deals with repairers, adjustors and assessors to execute these claims. Moreover, AGFIL has the capability to decide if both a given claim is valid against fraud and payment will be sent to the repairer. Below the main steps of our methodology.

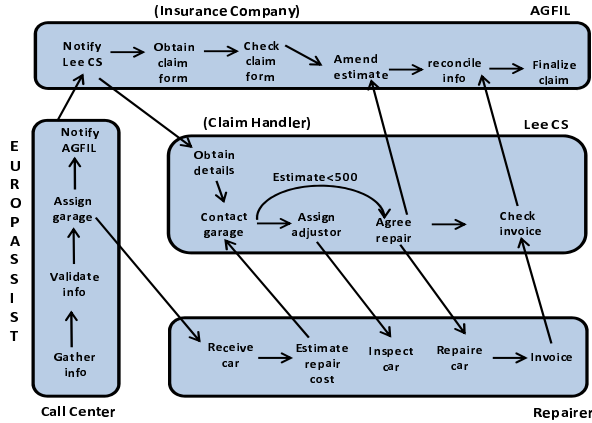


Fig. 3. Cross flow insurance claim processing [19].

A. Step 1

According to the ontology associated with the community, the functionality of the community is *InsuranceClaimProcessing* and this community includes all agent-based web services supporting the similar or part of this functionality. Thereafter, the engineering designer selects one of them as *master-ws₁* (as the manager of the company) that starts to cluster the slave web services into alliances structure based on the policy classes (a form of nonfunctional properties related to QoS), such as *class₁*, *class₂* and *class₃* cover 100%, 90%, 80% respectively from each claim request. We here only consider alliance structure that covers $QoS = \{100\%$ of the automobile damage and from now we refer to AGFIL by the Insurer (i.e., the role of this alliance is identified by the Insurer). The unique role names of slave web services that populate this alliance are defined as *call center* for Europ Assist and *assessor* for Lee CS, as well as *repairer* and *adjustor*. We modify this use case [16] by establishing a direct dependency between the Insurer on one side and repairer and adjustor on the other side. Also, we introduce the direct dependency between the customer and Insurer. At the end of this step, the Insure context is denoted by Ins_x and the *manager* (or *master-ws₁*) delegates the customer's request to the Insurer.

B. Step 2

The customer has one relevant goal: *vehicle repair*, while the Insurer has the goals: *handle claim* and *maximize*

profits. The last goal is represented in terms of soft goal. Thus the customer depends upon the Insurer to handle claim with $QoS = \{100\%$ and in exchange the Insurer depends upon the customer for paying the insurance premium [15]. Let us focus on such a goal, namely handle claim. The Insurer uses AND decomposition to decompose the goal ($G = \{handle\ claim\}$) into five subgoals, $G = \{claim\ reception, claim\ assessment, vehicle\ repair, claim\ finalization, vehicle\ inspection\}$. The Insurer delegates these subgoals to the respective roles within its alliance structure where the call center is responsible for the claim reception, the assessor for the claim assessment, the repairer for the vehicle repair and the adjustor for the vehicle inspection. The claim finalization will be performed by the assessor himself. By so doing, the Insurer pays service charge to each one of them after completing their goals.

C. Step 3

For the space limit reasons, we concentrate only on the call center, assessor and repairer task dependencies. The means-end analysis is used to identify tasks and task dependencies to each goal dependency from step (2). The claim reception goal, g_1 , of the call center depends on four tasks to achieve it, $t_1 = \{gathering\ info, validating\ claim, assigning\ garage, sending\ claim\}$. The call center gathers information and assigns a garage when the customer reports an accident and validates claim information. Thus, the repairer depends on the call center to assign a garage and the customer depends on call center for gathering claim information. Meanwhile, the validate claim information task is decomposed into two subtasks, $t_{1,2} = \{request\ policy\ information, validate\ information\}$. According to the architecture design the call center can define a new goal to receive payment of claim reception charge from the Insurer via executing a task of *receiving payment* or delegate the task to another (e.g., the reporter to prepare its report). Finally, it sends a valid claim to the Insurer to *finalize claim processing*.

The assessor has claim assessment goal, g_2 , delegated from the Insurer. The tasks needed to satisfy this goal are: $t_2 = \{receive\ claim, check\ invoice, agree\ to\ repair, obtain\ repair\ estimate, inspect\ vehicle\}$. The *receive claim* task depends upon *send claim* task of the Insurer to the assessor. Meanwhile, the Insurer and repairer depend on the assessors' tasks for *checking the invoice* and *agreeing to repair*. Moreover, the assessor depends upon the repairer to *obtain the repair estimate* by performing *estimation repair cost* task. The assessor depends upon the adjustor to *inspect a vehicle* and requires to define a new goal to *receive assessment fees* from the Insurer by executing a task of *receiving assessment fees*.

The repairer has a vehicle repair goal, g_3 , and the tasks needed to satisfy this goal are denoted by $t_3 = \{repair\ vehicle, estimate\ repair\ cost, send\ invoice\}$. The customer depends upon the repairer for *repairing vehicle* when received *valid claim* from the customer. The assessor depends upon the repairer for *estimating repair cost* to decide

agreeing to repair or negotiating with the repairer. Meanwhile, the repairer depends on the assessor for *checking the invoice* and forwarding it to the Insurer, if the repairer sends the invoice. Likewise, the call center and the assessor, the repairer requires to define a new goal to *receive repair charge* from the Insurer by depending on executing a task of *receiving repair charge* [6], [16].

D. Step 4

This step transfers each task dependency into an appropriate commitment to represent business meaning of interacting parties of AGFIL. In [16] the authors have ignored the formulation of actions on commitments and focused only on commitment itself (although actions on commitments reflect dynamic behaviors of agents). For example, commitment $C(id_1, call\ center, customer, Ins_x, report\ accident \wedge valid\ claim, assign\ garage)$ means that when the customer reports an accident and if the claim is valid, then the call center commits to assign a garage to him within the Insurer. But, how the Insurer delegates claim reception to the call center, how the Insurer formally assigns the assessor to get the inspection fees from the adjustor, how the call center withdraws from his commitment, etc. Here we complement [16] with commitment operations. From step (2) the Insurer has five subgoals (claim reception, claim assessment, vehicle repair, claim finalization and vehicle inspection) that are delegated to the call center, assessor, repairer and adjustor respectively. Formally we define this delegation operation, but in the case of the Insurer that delegates claim reception to the call center only as follows:

Delegate(*Insurer, call center, C(id₁, Insurer, Ins_x, customer, pay insurance premium, claim reception)*)

This operation intuitively means that the Insurer withdraws from the commitment and the call center creates a new commitment such that it becomes the debtor towards the customer to receive claim reception. Formally we need two steps to perform this:

Withdraw(*Insurer, C(id₁, Insurer, policy holder, Ins_x, pay insurance premium, claim reception)*) \wedge

Create(*call center, C(id_{1.1}, call center, customer, Ins_x, claim reception)*)

According to step (3) the claim reception needs four tasks to be achieved, one of them is gathering information from the customer. We define it formally as:

Create(*call center, customer, C(id_{1.1}, call center, customer, Ins_x, report accident, gather info)*)

Moreover, the Insurer assigns the assessor to obtain inspection fees from the adjustor when the estimate repair cost returned from repairer is more than 500 (a threshold amount).

Assign(*Insurer, assessor, C(id₂, adjustor, Insurer, Ins_s, pay inspection fees, estimate inspection cost)*)

This action is similar to the delegation action. Formally we need two steps to perform this:

Release(*Insurer, C(id₂, adjustor, Insurer, Ins_s, pay inspection fees, estimate inspection cost)*) \wedge

Create(*adjustor, C(id_{2.1}, assessor, Ins_x, pay inspection fees, estimate inspection cost)*)

E. Step 5

We define two subscenarios from AGFIL scenario to explain why we need argumentative dialogues to reason about the validity of commitment operations. The first subscenario is established between the Insurer and assessor where the assessor commits to the Insurer to reach agreement with the repairer for the vehicle repair, formally: $C(id_3, assessor, Insurer, Ins_x, pay\ assess\ fees, agree\ to\ repair)$ [16]. However, the assessor cannot estimate his assessment fees without engaging in a dialogue with the repairer to reach a deal about "estimate the repair cost". The solution proposed in [16], which is based only on commitments, is not enough to specify this dialogue especially when the assessor needs to negotiate the repair charge with the repairer. The proposed argumentative dialogues are natural solutions to this problem. The following dialogue game explains only the steps after the commitment being delegated to the repairer. This means that *request to estimate repair cost* (i.e., ψ) is supported by argumentative systems of the assessor and repairer (i.e., $\psi \triangleleft Arg_{sys}(assessor) \wedge \psi \triangleleft Arg_{sys}(repairer)$). Then the dialogue is opened and this dialogue can be considered as a continuation to example (2). The repairer creates a commitment towards the assessor to estimate the repair cost. The assessor has a conflict with the Insurer because the estimated cost does not respect the delegated constrains from the Insurer (estimate cost should be < 500 to maximize the Insurer profits). Then assessor challenges the repairer to justify the estimated repair cost. When the argumentation system of the assessor supports the justification of the estimate repair cost, then it reaches a deal with the repairer and informs the Insurer to pay the assessed fees.

- 1) **Create**(*repairer, C(id_{3.1}, repairer, assessor, Ins_x, ϕ)*) \wedge *Cond₁*
 \Rightarrow **Challenge**(*assessor, C(id_{3.1}, repairer, assessor, Ins_x, ϕ)*)
- 2) **Challenge**(*assessor, C(id_{3.1}, repairer, assessor, Ins_x, ϕ)*) \wedge *Cond₂*
 \Rightarrow **Justify**(*repairer, C(id_{3.2}, repairer, assessor, Ins_x, ϕ'), C(id_{3.1}, repairer, assessor, Ins_x, ϕ)*)
- 3) **Justify**(*repairer, C(id_{3.2}, repairer, assessor, Ins_x, ϕ'), C(id_{3.1}, repairer, assessor, Ins_x, ϕ)*) \wedge *Cond₃*
 \Rightarrow **Accept**(*assessor, C(id_{3.2}, repairer, assessor, Ins_x, ϕ')*)

Where :

$\phi = Estimate\text{-}Repair\text{-}Cost = V$, with V is a given value

$Cond_1 = \neg(\phi \triangleleft Arg_{sys}(assessor)) \wedge$

$\neg(\neg\phi \triangleleft Arg_{sys}(assessor))$,

$\phi' = \phi'_1 \wedge \phi'_2$, $Cond_2 = \phi' \triangleleft Arg_{sys}(repairer)$,

$Cond_3 = \phi' \triangleleft Arg_{sys}(assessor)$

Notice that the value of $\phi' = \phi'_1 \wedge \phi'_2$ means that the cost of repair includes the value of the *part₁* (ϕ'_1) and *part₂* (ϕ'_2).

Similarly, the adjustor and assessor can enter in negotiation dialogue about the estimated inspection cost. Moreover, argumentative dialogue in our case study can be used to spread agent-based web service knowledge in the form of information seeking dialogue. For example, when the call center requests policy information from the Insurer, the available information may be not enough to validate the received claim. Thus, the call center may need to spread his knowledge by requesting more information from the customer that enable the call center to send a valid information to the Insurer.

The second subscenario is established between the customer, Insurer and call center. When the customer reports an accident and his claim is valid, then the call center assigns a garage for vehicle repair. Otherwise, the call center does not assign a garage as it gathers invalid information from the customer compared to the information received from the Insurer. Thereby the intuitive semantic of the interaction between the call center and customer lacks clarity of business meaning related to the content of interaction (i.e, it is not meaningful). But, argumentative dialogue makes interaction more meaningful by guiding the call center to undertake a subtle decisions when the conflicts arise, then it challenges the customer to justify the validity of its commitment and to reach a mutual agreement that helps the customer to repair the vehicle. Moreover, the Insurer should persuade the call center, repairer, assessor and adjustor to join alliance structure by offering rewards that alliance grants to them.

V. RELATED WORK AND CONCLUSIONS

In this paper, we first exposed some obstacles that restrict the use of agent-based web services in complex business applications and then, offered some solutions to tackle these obstacles through extending community structure with alliances structure and proposing a new methodology based on Tropos methodology. This methodology synthesizes three approaches: mentalistic states, social commitments and argumentative dialogues. The mentalistic artifacts such as goals, tasks and dependencies modeling techniques are compatible with concepts of Tropos (e.g., plan in Tropos is the task here). The social commitments define dependencies between agent-based web services in the early stages as business-meaning of interactions and guide Tropos for accommodating and tracing the changes in the various phases of the methodology. While Tropos provides commitment with cues that identify different parties of dependencies (e.g., agents, goals, plans). Moreover, commitments enable the community to support dynamic reconfiguration of business interactions via delegation and assignment operations without altering the overarching structure of web services interactions. Argumentative dialogues support web services with richer capabilities to negotiate or seek information with other peers via reasoning about the correctness of commitments

The social-arguments proposed here are different from the approach proposed in [8] for argumentation-based negotiation based on social commitments. The agents are influenced by social relationships and negotiate with other peers based on

their organizational roles. The approach lists the possible rules that can be applied when conflicts of interest occur. These rules, for example, are used to reject or accept proposals as well as to enforce the social relations within a multiagent system. However, the rules in [8] are not used for justifying the content of commitments within negotiation dialogue and they do not consider agents' goals that can affect agents decisions as we have done here.

Let us now focus on comparing the proposed methodology with the related ones. Many of Agent-Oriented Software Engineering (AOSE) methodologies have been proposed over the last years based on the concepts of actors, roles, goals and plans include *Gaia*, *Prometheus*, *MaSE*, and *Tropos*. These methodologies support various phases of the software development life cycle, but Tropos differs from them in including an early requirements phase. Moreover, some of these methodologies like *Gaia* differs from Tropos in involving safety and liveness conditions for the processes and agents should be coarse-grained computational systems (like UNIX process). Meanwhile, *Gaia* lacks reasoning scheme based on early requirements engineering which limits the flexibility of *Gaia*, as well as implementation phase is not covered in this methodology. Our methodology complements these methodologies by concentrating on social arguments and argumentative dialogues, which they are ignored in these methodologies. A key difference between our methodology and Tropos [17] is the considerations of concepts of community, commitment and argumentative dialogue. In Tropos the depender depends upon the dependee for achieving a goal or executing a plan without any condition from the depender's side, but in commitment the debtor is obliged towards the creditor to bring about its commitment when a condition is hold. Thus, Tropos lacks capability to model real-life business scenarios between economic partners [15].

Pankaj et al. [15] enhanced Tropos methodology with concept of commitments to capture business meaning of interactions among independent parties in early requirement phases and successively refined using means-end and AND/OR analysis during the progress of the system being engineered. This methodology [15] is close to our methodology but it lacks the argumentative capabilities to enable agent-based web services to negotiate or seek information with other peers based on their commitments. Moreover, the authors translate goals into concrete tasks for their fulfillment before considering the overall organization of the system, which makes software systems fragile and less reusable although commitments can be modified. The reason is that they ignore architecture design phase. We present the architecture design phase with alliances structure to satisfy a number of quality requirements related to performance, usability, modifiability and reusability. For example, the Insurer can delegate the responsibility to the assessor to finalize users' claims or to pay inspection fees to the adjustor on his behalf.

Desai et al. [7] proposed Amoeba as a process modeling methodology based on commitment protocols. This methodology guides software designers to evolve requirements based

on separating service interactions into two layers: commitment protocols and policies. They represent business process in terms of fine-grained messages with commitments where commitments capture business interactions. Our methodology includes in addition of that mentalistic states and argumentative dialogues.

Riemsdijk et al. [18] used a goal-oriented approach inspired by the field of cognitive agent programming for service-oriented computing to help handling failures and specify the semantics of the services. An orchestration approach has been used to coordinate the invocations of services from a workflow. Burmeister et al. [5] presented an approach to business process management using BDI-agent features to capture the ability of the process to adapt and pro-actively adapt itself to a changing environment (or what they call an “agile process”) to avoid problems before they arise. The goal-oriented approach is the core of our methodology, but it differs from [5], [18] in considering goals as commitments, which provide more flexibility in terms of manipulation (e.g., assign, release or withdraw) and agents use argumentative dialogues to enable each participant to satisfy its hard goals and soft goals in an efficient manner. Moreover, an orchestration reflects only one participant’s view of the overall business process and lacks business meaning of service engagements. The conceptual framework proposed in [5] has been successfully applied to a business process for engineering change management domain, but it lacks the cooperation between agents.

Li et al. [9] proposed an agent-based framework to model and develop dynamic service-oriented operations. In our approach, web services are also viewed as software agents, but what is new in our approach is that web services can communicate with other peers within a community by dialogue game protocols.

Regarding to CWSs, Maamar et al. [11] recently presented an engineering methodology for modeling CWSs based on the concepts that assist in using community, selecting web services, identifying allowable operations to each web service and deploying community. However, they ignore the collaboration between web services, high-level business meaning of interactions, alliance structure, user interactions (although these interactions are crucial and ought to be recognized within community). Last but not least, Medjahed et al. [14] proposed the WebBIS system as a generic framework for composing and managing web services in terms of pull-and push-communities within dynamic environments. Our methodology underpins a simple mechanism for empowering composition of web services based on means-end and AND/OR analysis within the notion of delegation operation that captures flexible interactions.

As future work, we aim to formalize the strategic policies of service providers that help organizing web services in alliances structure and investigating the organization laws and norms ruling the roles within the community. The trustworthiness level of a master web service towards alliance web services and how a social-argumentative dialogue model is automatically mapped into a BDI-agent specification that can,

at execution time, give useful feedback to refine the original design are fundamental issues we plan to investigate.

ACKNOWLEDGEMENTS

The authors would like to thank the reviewers for their valuable comments and suggestions. They also would like to thank NSERC (Canada), NATEQ FQRSC (Québec) and ERESON for their financial support.

REFERENCES

- [1] B. Benatallah, M. Dumas and Q.Z. Sheng. Facilitating the Rapid Development and Scalable Orchestration of Composite Web Services. *J Distrib Parallel Databases* vol.17(1), pp.5-37, 2005.
- [2] B. Benatallah, Q.Z. Sheng and M. Dumas. The Self-Serv Environment for Web Services Composition. *IEEE Internet Computing*, vol.7(1), pp.40-48, 2003.
- [3] J. Bentahar, Z. Maamar, D. Benslimane and P. Thiran. An Argumentation Framework for Communities of Web Services. *IEEE Intel. Sys.* vol.22(6), pp.75-83, 2007.
- [4] J. Bentahar, Z. Maamar, W. Wan, D. Benslimane, P. Thiran and S. Subramanian. Agent-Based Communities of Web Services: An Argumentation-Driven Approach. In *Service Oriented Computing and Applications*, Vol.2(4), pp.219-238, 2008, Springer.
- [5] B. Burmeister, M. Arnold, F. Copaciu, G. Rimassa. BDI-Agents for Agile Goal-Oriented Business Processes. In *Proc. of 7th Int. Conf. on Aut. Agents and Multiagent Sys. (AAMAS 2008)*, pp.37-44, 2008.
- [6] N. Desai, A.K. Chopra and M.P. Singh. Business Process Adaptations via Protocols. In *Proc. of the IEEE Int. Conf. on Services Computing (SCC)*, pp.103-110, 2006.
- [7] N. Desai, A.K. Chopra and M.P. Singh. Amoeba: A Methodology for Modeling and Evolution of Cross-Organizational Business Processes. *ACM Transactions on Software Eng. and Methodology (TOSEM)*, 2009.
- [8] N.C. Karunatillake, N.R. Jennings, I. Rahwan and T.J. Norman. Argument-Based Negotiation in a Social Context. In *Proc. AAMAS Workshop on Argumentation*, pp.74-88, May 2005.
- [9] Y. Li, W. Shen and H. Chenniwa. Agent-Based Web Services Framework and Development Environment. *Comput Intell*, vol.20(4), 2004.
- [10] Z. Maamar, M. Lahkim, D. Benslimane, P. Thiran and S. Sattanathan. Web Services Communities-Concepts and Operations. In *Proc. of the 3rd int. conf. on web information sys. and technologies (WEBIST'2007)*, Barcelona.
- [11] Z. Maamar, S. Subramanian, J. Bentahar, P. Thiran P and D. Benslimane. An Approach to Engineer Communities of Web Services Concepts, Architecture, Operation, and Deployment. In *the Int. Journal of E-Business Research*, vol.5(4), 2009, IGI Global.
- [12] B. Medjahed and Y. Atif. Context-Based Matching for Web Service Composition. *Distrib Parallel Databases*. Springer, Heidelberg, vol.21(1), pp.5-37, 2007.
- [13] B. Medjahed and B. Bouguettaya. A Dynamic Foundational Architecture for Semantic Web Services. *Distributed and Parallel Databases*. Kluwer, Dordrecht, vol.17(2), pp.179-206, 2005.
- [14] B. Medjahed, B. Bouguettaya and A. Elmagarmid. WebBIS: An Infrastructure for Agile Integration of Web Services. *Int. J. Cooperative Inf. Syst. (IJCS)*, vol.13(2), pp.121-158, 2004.
- [15] P.R. Telang and M.P. Singh. Enhancing Tropos with Commitments: A business Metamodel and Methodology. Alex Borgida, Vinay Chaudhri, Paolo Giorgini and Eric Yu (eds), *Conceptual Modeling: Foundations and Applications*, June 2009.
- [16] P.R. Telang and M.P. Singh. Business Modeling via Commitments. In *Proc. of the 7th AAMAS Workshop on Service-Oriented Computing: Agents, Semantics and Eng. (SOCASE)*, May 2009.
- [17] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia and J. Mylopoulos. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems*, vol.8(3), pp.203-236, 2004.
- [18] M.B. van Riemsdijk, M. Wirsing. Using Goals for Flexible Service Orchestration: A First Step. In *Service-Oriented Computing: Agents, Semantics, and Eng.* vol.(4504) of LNCS, pp. 31-48, 2007, Springer.
- [19] S. Browne and M. Kellett. Insurance (Motor Damage Claims) Scenario. Document Identifier D1.a, CrossFlow Consortium, 1999.
- [20] M.P. Singh. An Ontology for Commitments in Multiagent Systems: Toward A unification of Normative Concepts. *AI and Law*, vol.7, pp.97-113, 1999.

Programming SOA/WS Systems with BDI Agents and Artifact-Based Environments

Michele Piunti
DEIS, University of Bologna
Cesena, Italy
Email: michele.piunti@unibo.it

Andrea Santi
DEIS, University of Bologna
Cesena, Italy
Email: andrea.santi6@studio.unibo.it

Alessandro Ricci
DEIS, University of Bologna
Cesena, Italy
Email: a.ricci@unibo.it

Abstract—Agents and Multi-Agent Systems are recognized in the literature as a suitable paradigm for engineering SOA and Web Service systems: however few works explore how to exploit agent programming languages – in particular those based on a strong notion of agency, such as BDI ones – for concretely developing such a kind of systems. In this paper we discuss a general-purpose programming model and a related platform for developing SOA/WS applications exploiting BDI agent technologies. In particular, in order to enable agents to exploit and manage web service technologies in a suitable functional fashion, we investigate the use of *Jason* agents – based on AgentSpeak(L) programming language – integrated with artifact-based environments – based on CArtAgO-WS framework.

I. INTRODUCTION

Agents and Multi-Agent Systems are more and more recognized in the literature as a suitable paradigm for engineering SOA and Web Service systems, since they provide a conceptual and engineering background that naturally fits many complexities concerning SOA/WS at a high abstraction level [13], [14], [10]. Actually this view is also promoted both by the official service-oriented model described by W3C (<http://www.w3.org/TR/ws-arch/>) and by the OMG initiative about the definition of an agent meta-model and profile in the SOA perspective (<http://www.omg.org/cgi-bin/doc?ad/2008-09-05>).

In this perspective, besides being an effective meta-model to *design* SOA, we argue that the agent-oriented programming languages and technologies can be effective tools for concretely *programming* SOA and Web Services applications, in particular for those kinds of service-oriented systems that need to integrate advanced features such as autonomy, flexibility, reactivity, asynchronous interaction management [6], [14]. Accordingly, in this paper we develop this issue by presenting and discussing an approach which exploits and integrates existing agent technologies – *Jason* agent programming language [3] and CArtAgO-WS framework [20] – into a general-purpose platform to program and execute SOA/WS applications. In particular, the approach allows for programming and running SOA/WS applications as multi-agent systems composed by agents based on Belief Desire Intention (BDI) working together in shared environments. Besides, agent working environments are instrumented with specific kinds of tools (namely, artifacts) that agent can use to interact with existing Web Services (as consumers), to implement Web Services (as providers) and to exploit higher-

level service-oriented capabilities, such as WS-Coordination. In the proposed model artifacts are special computational entities providing the access point to Web Services, they can be created and configured on the need and are exploitable in a functional / goal-oriented fashion in order to build and consume complex SOA applications.

Several frameworks have been presented in the agent area for the design of SOA. Actually they mainly focus on the *integration* of agent platforms – in particular, FIPA-based platforms, such as JADE – with Web Services technologies [11], [15], [25]: their design objective is mainly to find a common specification to describe how to seamlessly interconnect FIPA-compliant agent systems with W3C-compliant Web Services. The proposed solutions usually adopt some kind of centralized *gateway*, working as a mediator for agents who aim to interact with Web Services on the one side (agents as service consumers) and for Web Service requests to be served by agents on the other side (agents as service provider) [11]. Conversely, the approach presented in this work is based on a dynamic creation and control of customized artifact-based facilities aimed at supporting agent activities at an infrastructural level. We argue that this would improve the modularity, scalability and (dynamic) extensibility of the systems.

Besides enabling interoperability between agents platforms and web services, a main objective of this work is to investigate the use of a strong notion of agency – and in particular agent programming languages supporting it – exploiting artifact-based environments to concretely design and build service-oriented systems. In this view we promote the integration of both task-oriented/process-oriented behavior – such in the case of agent based workflows [1] or goal-oriented business processes [24] – and a reactive (even-driven) behavior, such in the case of Event-Driven Architectures (EDA), which are meant to be a main aspect of forthcoming SOA. In this perspective, our work is related to existing approaches investigating the use of goal-oriented/BDI agent technologies in the context of Web Services (see, among others [5], [4], [9], [26]). The specific focus on strong agency is also the main novelty of this paper with respect our previous work [20], [19], where a SOA/WS programming model based on A&A and the related platform have been introduced.

The remainder of the paper is organized as follows: in

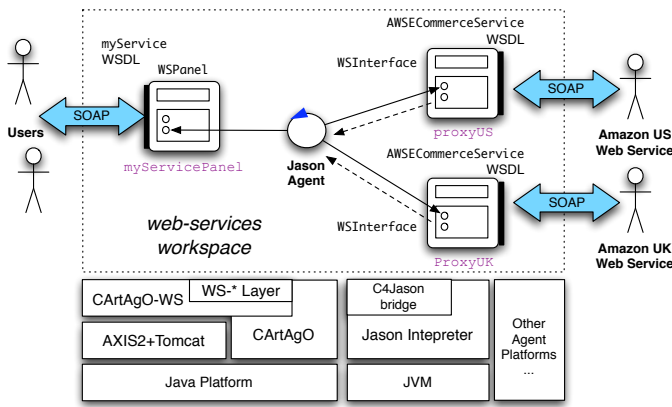


Fig. 1. CArTAgO-WS platform overview. The figure shows a CArTAgO-WS node running a Web Service built with a single agent and some artifacts deployed in the *web-services* workspace: on the right, two instances of *WSInterface* artifact are used by an agent to interact (concurrently) with two external Web Services; on the left, the agent uses a *WSPanel* artifact to provide itself a service, serving the requests coming from external WS users. In the bottom, the layers and technologies on top of which the application is built are shown.

Section II we briefly describe the basic concepts and technology on which the programming model and platform is based; in Section III a case study is discussed, where pivotal features of BDI agents as autonomy, flexibility, proactiveness and reactivity are emphasized in a web service application involving the use of WS transactions and protocols.

II. PROGRAMMING MODEL AND CArTAgO-WS PLATFORM

This section provides a global picture of programming model based on BDI agents and artifacts for implementing Web Services and SOA. In the next sections CArTAgO-WS platform is described, along with a simple example showing the approach in practice.

A. Agent and Artifact Programming Model for Web Services

The proposed programming model for implementing Web Services relies on A&A (Agents and Artifacts) meta-model, recently introduced in the context of agent-oriented software engineering [16]. In A&A perspective, a service – or an application using services, that can be a service itself – is organized in terms of a set of agents – as autonomous, pro-active entities – that work together inside a shared computational environment, properly designed to support their activities.

Such computational environment – possibly distributed across several nodes – is organized in terms of *workspaces* containing sets of first-class entities, called *artifacts*, representing tools and, more generally, resources that agents share and use to cooperate and fulfill their tasks. So artifacts are the basic abstraction that MAS designers and programmers can exploit to conceive and program agent environments, encapsulating functionalities that – at runtime – agents can exploit to *externalize* tasks and thus achieve their (individual and collective) objectives.

A detailed description of agents and artifacts programming model is outside the scope of this work (the interested readers can find more details in previous papers [21], [22]). Here we

exploit agents and artifacts as means to design and program a SOA/WS application as a multi-agent system, in particular as workspaces where goal-oriented agents work together sharing and exploiting environment-based facilities:

- *Agents* are meant to encapsulate the logic and control of tasks, activities and business processes – both in the case of client applications and service applications;
- *Artifacts* are used to represent specialized resources and tools inside the workspaces that agents can exploit, useful in particular – in this case – to encapsulate and hide low-level aspects related to WS management.

In what follows, an integrated programming model based on BDI agents and artifacts is discussed, firstly with respect of environment setting and then with regard of agent development. In particular, *Jason* is adopted as reference programming platform for BDI agents¹. Besides, CArTAgO-WS² (Common ARTifact infrastructure for AGent Open environment and Web Services) is adopted as environment programming platform.

B. Environment Side

CArTAgO-WS has been recently introduced as the reference technology for implementing SOA working environments based on the A&A model [20]. The platform integrates different modules supporting, from the one side, agent based frameworks and, on the other side, a seamless integration with Web Service technologies. As depicted in Fig. 1 (*bottom*), CArTAgO-WS is currently implemented on top of existing open-source WS technologies as Axis2 (see <http://ws.apache.org/axis2/>), in order to conform to the Basic Profile specification of Web Service Interoperability Organization (WS-I). The core technology of CArTAgO-WS is CArTAgO [22], which provides both a concrete computational/programming model for developing and running artifact-based environments, and API to integrate existing agent technologies (and languages, architectures) with it. This enables the implementation of MAS populated by agents possibly developed using different agent languages working together inside the same artifact-based environment.

In CArTAgO, artifacts are characterised by a *usage interface* listing a set of controls that agents can use to trigger and execute artifact *operations*, structuring artifact functionalities. Operation execution can lead to the generation of *observable events* that the agent using the artifact and other agents possibly observing it can perceive.

Basically CArTAgO-WS extends CArTAgO by providing a predefined workspace called *web-services*: this workspace is dynamically instrumented with different kind of specialized artifacts aimed at working with Web Services:

- *Basic artifacts*, aimed at enabling basic interactions between agents and Web Services;

¹*Jason* is an open source platform for programming BDI agents based on AgentSpeak(L). For brevity, we omit the description of the syntax and semantic of the language: the interested reader can find more in [2] and <http://jason.sourceforge.net>.

²CArTAgO-WS is an open source platform available at: <http://cartago.sourceforge.net>.

- *WS-* artifacts*, aimed at supporting an enriched set of interactions, as the ones envisaged by the Web Services stack protocol;
- *Business artifacts*, aimed at providing functions for supporting agents in their business activities, as storing information which is relevant for the ongoing task in a database, wrap an external resource, control a user interface etc.

In what follows a description of the artifacts holding on the first two groups is provided, while an example of artifacts holding to the third group is given in the application of Section III.

Artifacts of the basic group allow, on the one side, agents to work with existing Web Services and, on the other side, allow the construction and the deployment of new Web Services controlled by agents. In particular, two configurable artifacts introduced are `WSInterface` and `WSPanel` artifacts (Fig. 1 shows an example of their use). To interact with an existing Web service, an agent instantiates a `WSInterface` artifact specifying its WSDL document which describes the service to interact with. Optionally it takes in other parameters such as the specific service name/port type to be used (if the WSDL includes multiple port types and services), and a local name representing the endpoint to which the artifact is bound to receive messages (e.g. replies). Once created, `WSInterface` provides basic functionalities to interact with the specified Web Service, such as sending a message to the service in the context of an operation (`sendWSMsg` usage interface control) or getting the reply to messages previously sent during an operation (`getWSReply`). Besides, it includes higher-level operations to directly support basic MEPs, such as the request-response (in-out) MEP (`requestOp`) which sends a request message and generates an event when the response message arrives.

Current implementation makes use of SOAP messages for executing operations and to get the replies sent back by the service, according to the message exchange patterns defined in the WSDL and to the quality of service specified by the service policies (in particular, security and reliability). In future implementation of this artifact we plan to support also resource-oriented interaction with services, as promoted by the REST architectural style [8].

To use multiple Web Services, multiple `WSInterface` artifacts must be created, one for each service: agents can then use such artifacts to interact with the services concurrently. Different agents can also use the same `WSInterface` artifact to interact with the same service.

For creating, configuring and controlling a new Web Service, a `WSPanel` artifact is provided. Analogously to the previous case, `WSPanel` can be instantiated specifying a WSDL document. Once created, `WSPanel` provides basic functionalities to manage SOAP requests, including receiving and sending messages according to the specific MEP as described in the WSDL, and basic controls to configure security and reliability policies. Also in the case of `WSPanel`, the usage interface includes a set of general purpose operations enabling the interaction according to the wide spectrum of

possible WS messaging patterns. Operations are available to retrieve or be notified about requests/messages arrived to the Web Service possibly specifying filters to select messages on the basis of their content/meta-data (`getWSMsg`, `getWSMsgs` and `subscribeWSMsgs`) and to send replies accordingly (`sendWSReply`).

It is worth remarking that agents can *dynamically* create, quit and re-create both `WSPanel` and `WSInterface` once they have joined a `web-services` workspace hosted in a `CARTAgO-WS` node: this allows to dynamically deploy and re-configure Web Services not by human intervention but by agents activities, thus promoting an automated management of services. Accordingly, it is possible to instantiate or interact with multiple Web Services at the same time, i.e. by creating multiple `WSPanel/WSInterface` artifacts, one for each service.

Besides the basic interactions promoted by the above mentioned artifacts `CARTAgO-WS` introduces an additional group of artifacts. This group is included in the `WS-*` layer of the platform (see Fig. 1) and is aimed at supporting an extensible set of WS specifications, in particular those appearing in the WSIT (Web Services Interoperability Technologies) set (see <http://wsit.dev.java.net>). For doing this, the `WS-*` layer is instrumented with two kinds of specialized artifacts: the `WSRequestMediator` and the `Wallet`.

The `WSRequestMediator` (RM) artifact is meant to be used by agents to retrieve (or create) those dynamic information required by complex specification such as WS-Coordination (WS-C). RM's provides a general purpose usage interface so that multiple RM can be instantiated to conform to multiple protocols. For instance, suppose that an agent aims at creating a new WS-AtomicTransaction (WS-AT): to this end, an agent can use a RM to (create and) retrieve a specific coordination context, which has been previously configured following WS-Coordination and WS-AT standards.

Besides RM, a `Wallet` artifact is introduced as "personal artifact" that agents interacting with Web Services can exploit to support the management of profile/context information eventually needed by WS specification. The `Wallet` works in synergy with RM artifacts and its function is to dynamically store a portfolio of various policies which are required to conform messages to `WS-*` protocols. This information can range from security tokens (as required by WS-Security) to dynamic coordination contexts (as used in WS-C). In so doing a user agent can completely externalize on the wallet the management of the required policies. In a typical scenario, a agent using a Web Service first gets profile information from the `Wallet` and then uses it to configure the `WSInterface`.

So in the overall the `WS-*` layer allows MAS programmers to build articulated WS applications abstracting as much as possible from low-level details that concern `WS-*` specific protocols management (e.g. the management of the coordination contexts in WS-C), and to focus on the high-level functionalities (e.g. transactions) that agents may need to setup/exploit.

C. Agent Side

As mentioned in Subsection II-A, in our approach agents are the computational entities to specify and program the business logic of services (or applications using services). In particular, in this paper we argue that the use of a BDI agent based specification improves the abstraction by which a programmer can specify a complex task (i.e. a business process) in terms of structured agents' behavior. As agents programmable according to a BDI style can be specified by goal-oriented languages – in this paper we will use *Jason*, in particular, but analogous considerations hold by considering other agent programming languages/platforms such as 2APL [7], Jadex [18], etc. – the programming style for specifying business processes conceived as a complex chain of interleaved tasks can be natively conceived in a goal-oriented format, and thus expressed in terms of agent's plans. Then, as agents execution model is typically based on practical reasoning, BDI agents are highly adaptive in suitably finding a proper course of actions to achieve a given goal in the situated context conditions. In so doing, a main concern for programmers is to simply specify a set of behaviors realizing agent's tasks in terms of goals and plans.

Besides practical reasoning, an additional remarkable aspect is the interaction model defining interaction between agents and artifacts. From an agent point of view, artifact computational model allows two kind of interaction, as they are based on the notions of *use* and *perception*. Given this, as agents exploit artifacts to provide or use services, their programs can be expressed in native terms, i.e. by the mean of primitives for actions and perceptions. An additional remarkable aspect is related to the perceptive abilities carried out by agents with respect to artifact observable events. Indeed, due to the computational model provided by CArtAgO-WS, agents can be highly sensitive towards a rich series of events occurring upon a focused artifacts. Due to the fact that BDI model of agency typically provides constructs to explicitly handle noticeable events and react accordingly, a clear definition of events in terms of agent percepts allows a situated reactivity of agents, mainly addressed towards the perception of relevant changes affecting the work environment, (where for relevant we refer to those information which is assumed to support the ongoing plans, as goal supporting beliefs).

To provide a concrete taste of the programming approach, the structure of a simple application is showed in Fig. 1, accounting for a service finding the best price of items – books in the specific case – by interacting with two existing web services (two Amazon Web Services, one for UK and one for US)³. The service is implemented by a single Jason agent, using a *WSPanel* artifact to retrieve service requests (and send responses) and two *WSInterface* artifacts to interact with the two existing Web Services.

A cutout of the agent source code is shown in TABLE I,

³The complete source code of the examples as well as the WSDL of the implemented services are available at CArtAgO-WS web site: <http://cartago.sourceforge.net>.

```

01 !find_best_price_service.
02
03 +!find_best_price_service
04 <- cartago.joinWorkspace("web-services","localhost");
05 !setupTools;
06 !cartago.use(wsPanel,subscribeWSMsgs("GetBestPrice")).
07
08 +ws_msg(Msg)
09 <- !extractMsgId(Msg,ReqId);
10 !extractItem(Msg,Item);
11 +pending_request(ReqId,Item,Msg).
12
13 +pending_request(ReqId,Item,Msg)
14 <- !prepareItemReqWs(Item,MsgReq);
15 cartago.use(awsUS,requestOp("ItemSearch",MsgReq));
16 cartago.use(awsUK,requestOp("ItemSearch",MsgReq)).
17
18 +ws_reply(ReplyMsg,ReqId)[source(From)] :
19 pending_request(ReqId,Item,Msg)
20 <- !extractPrice(ReplyMsg,Price);
21 +price(ReqId,Item,From,Price).
22
23 +price(ReqId,Item,_,_) : price(ReqId,Item,"awsUK",_) &
24 price(ReqId,Item,"awsUS",_)
25 <- !prepareAndSendResponse(ReqId).
26
27 +!prepareAndSendResponse(ReqId)
28 <- !computeBestPrice(ReqId,Price,From);
29 !prepareAmazonWSReply(Price,From,ReplyMsg);
30 cartago.use(wsPanel,sendWSReply(Msg,ReplyMsg));
31 -pending_request(ReqId,_,_);
32 -price(ReqId,_, "awsUK",_);
33 -price(ReqId,_, "awsUS",_).
34
35 +!setupTools
36 <- cartago.makeArtifact(wsPanel,"alice.cartagows.WSPanel",
37 ["/data/BestPriceService.wsdl"]);
38 cartago.makeArtifact(awsUS,"alice.cartagows.WSInterface",
39 ["/../AWSECommerceService.wsdl"]);
40 cartago.makeArtifact(awsUK,"alice.cartagows.WSInterface",
41 ["/../UK/AWSECommerceService.wsdl"]);

```

TABLE I
CUTOUT OF THE JASON AGENT SHOWN IN FIG. 1

which is briefly described in the following. The agent has a single initial goal (*find_best_price_service*, specified at line 01) and a set of plans that describe how to achieve this goal and related sub-goals. The first plan (line 03–06) is triggered as soon as the goal is instantiated, and accounts for setting up the tools needed to do the job (sub-goal *setupTools*, which triggers the execution of a plan (lines 35–41) creating the service panel, referenced by the atom *wsPanel*, and of the two WS interface artifacts, referenced as *awsUK* and *awsUS*) and subscribing the panel (operation *subscribeWSMsgs*) to receive all the message requests arriving to the service concerning the *GetBestPrice* operation.

As soon as a new WS request arrives to the panel, it generates an observable event *ws_msg(Msg)*. The agent reacts to the perception of that event (plan at lines 08–11) by extracting information about the message identifier and item to search (sub-goals *extractMsgId* and *extractItem* at line 09-10, not reported here for simplicity) and creating a *pending_request* belief containing information about the new request to process (line 11). The addition of new *pending_request* beliefs triggers the execution of a specific plan to process the requests (line 13–16). The plan accounts for using the *awsUK* and *awsUS* artifacts (lines 15 and 16) to request information about the item. The two services will answer asynchronously, with messages that are translated by the *WSInterface* in *+ws_reply(Resp,MsgId)* percepts. As

soon as replies arrive, the agent creates new price beliefs carrying information about the price of the item sold by the specific sources (plan at line 18–21).

As soon as both the price information from the UK service and US service are available for a specific request, the agent can prepare and send the response (sub-goal `prepareAndSendResponse`, line 25). This is done by the plan listed at lines 27–33, in which the best price is computed by the sub-goal `computeBestPrice` (not showed) exploiting the information stored in price beliefs and a reply message (related to the original message request) with the answer is sent back through the service panel (line 30). Finally, information about the specific request identifier and related prices are removed from the belief base (line 31–33).

Despite its simplicity, the example is meant to show how the approach allows for structuring quite naturally the business logic of the service in terms of agent plans, both to react to events occurring in artifacts populating the workspace (a `WSPanel` and the two `WSInterface` in the example) and to pro-actively execute sub-tasks on which the service business process is decomposed. By properly externalizing functionalities in artifacts, the approach makes it possible on the one side to keep agent behaviour relatively clean, purely focussed on the specification of the service logic, and, on the other side, to fruitfully exploit concurrency – artifacts execute operations in separated threads of controls – without dealing (for the agent programmer) with low-level synchronization issues. Finally, the approach promotes modularity and scalability. In the example a single agent is used to process and serve all requests arriving to the service: alternatively, a pool of agents can be used for this purpose, sharing the panel to get the requests.

III. PROGRAMMING COMPLEX WEB SERVICES USING BDI AGENTS AND ARTIFACT-BASED ENVIRONMENTS

The benefits of adopting a BDI model of agency along with artifact-based environments are evident in particular when the design and development of complex service applications are of concerns. In what follows, Subsection III-A introduces an example application involving some of the motivating elements at the basis of the proposed approach, while Subsection III-B discuss an implementation based on `CArtAgO-WS` and `Jason`.

A. A Case Study: Book an Holiday Scenario

The described scenario is inspired by a typical example used in SOA/WS contexts: a client agent wants to book an holiday for a given date by exploiting a series of web services providing the required resources as hotel reservation, transport facilities, payment and so on. As an additional element of the scenario, we imagine for the client the possibility to be further notified whether a selected range of date has become available for additional reservations. This allows clients to express an interest for a given date, and thus to re-try the booking activity whether the provider signals a last minute availability (i.e. due to some reservation cancelation performed

```

00  +!start_booking
01  <- !setupTools;
02  !retrieveDate;
03  !book_an_holiday.
04
05  +!setupTools : true
06  <- !locate_artifacts;
07  // Use the RM to request a new WS-AT and
08  // add the related ATContext into the Wallet
09  cartago.use(Wallet, addInfo(ATContext));
10  !makeInterface(proxyHM,
11  "http://webservices.hotel.com/.../BookingManager.wsdl");
12  ?artifact_id(proxyHM, ProxyID);
13  cartago.use(ProxyID, configure(ATContext)).
14
15  /* Top Level Goal */
16  +!book_an_holiday
17  : date(Dates)
18  <- !book_hotel(Dates, Res_H);
19  !book_accessories(Dates, Res_A);
20  !finalize(Res_H, Res_A).
21
22  +!book_hotel(Dates, Res_A)
23  : artifact_id(proxyHM, ProxyID)
24  <- !createBookingMessage(hotelBooking, Dates, MsgBookHotel);
25  cartago.doRequestResponse(ProxyID,
26  bookingOperation(MsgBookHotel), HotelResponse);
27  !inspect_h_response(HotelResponse, Res_H);
28  Res_H == "available". // fail if not available
29
30  +!book_accessories(Dates, Res_H)
31  : artifact_id(proxyTransport, TranID) & artifact_id(proxyPayment, PayID)
32  & hPrice(HotelPrice) & tPrice(TransportPrice) & bank_account_id(BankID)
33  <- !createBookingMessage(transportBooking, Dates, MsgTransport);
34  cartago.doRequestResponse(TranID,
35  bookingOperation(MsgTransport), ResponseTransport);
36  !createPayMessage(BankID, (HotelPrice+TransportPrice), MsgPay);
37  cartago.doRequestResponse(PayID,
38  payOperation(MsgPay), ResponsePayment);
39  !inspect_acc_responses(TransportResponse, PaymentResponse, Res_A);
40  Res_A == "available". // fail if not available
41
42  /* Fail Event Handling */
43  -!book_an_holiday
44  : artifact_id(proxyHM, ProxyID) & dates(Dates)
45  <- !createSubscribeMessage(Dates, MsgSubscription);
46  cartago.focus(ProxyID);
47  cartago.use(ProxyID, subscribeOperation(MsgSubscription));
48  !finalize("not_available", "").
49
50  /* Notification from HM */
51  +dateNotMoreFull(Dates) [source(proxyHM)]
52  : artifact_id(proxyHM, ProxyID) & dates(Dates)
53  <- cartago.stopFocusing(ProxyID);
54  !book_an_holiday;
55
56  /* Finalize */
57  +!finalize(Res_H, Res_A)
58  : Res_H == "available" & Res_H == "available"
59  & wallet_entry(wsatcontext, ATContext) & artifact_id(wsProxyCoord, CoordID)
60  <- !createCommitMessage(WS-AT-Context, MsgCommit);
61  cartago.doOneWay(CoordID, commitOperation(MsgCommit));
62
63  +!finalize(Res_H, Res_A)
64  : (Res_H /= "available" | Res_A /= "available")
65  & wallet_entry(wsatcontext, ATContext) & artifact_id(wsProxyCoord, CoordID)
66  <- createRollbackMessage(ATContext, MsgRollback);
67  cartago.doOneWay(CoordID, rollbackOperation(MsgRollback)).

```

TABLE II

Jason CUTOUT OF THE BOOKING REQUESTOR AGENT SHOWN IN FIG. 2

by other clients). On these basis, the involved services need to shape their activities based on situated conditions:

- A given transaction can have success, or not, given the resources which are *actually* available.
- The same transaction can be retried, based on changed contexts for which, at the moment of the first attempt, the provider could not finalize the task.

To achieve such a flexibility, service behavior can be straightforwardly expressed in terms of goal-oriented agents, where goals are expressed in terms of specific task to achieve (i.e. to book an holiday, to provide reservations, etc.). To achieve their goals agents can organize their workflow in terms of situated plans, involving the interaction with heterogeneous resources (such as internal resources as databases, coordination and transaction facilities, other web services, etc.). Accordingly, we will design and program the involved services based on BDI (goal-oriented) agents programmed in `Jason` exploiting a `CArtAgO-WS` web-services workspace.

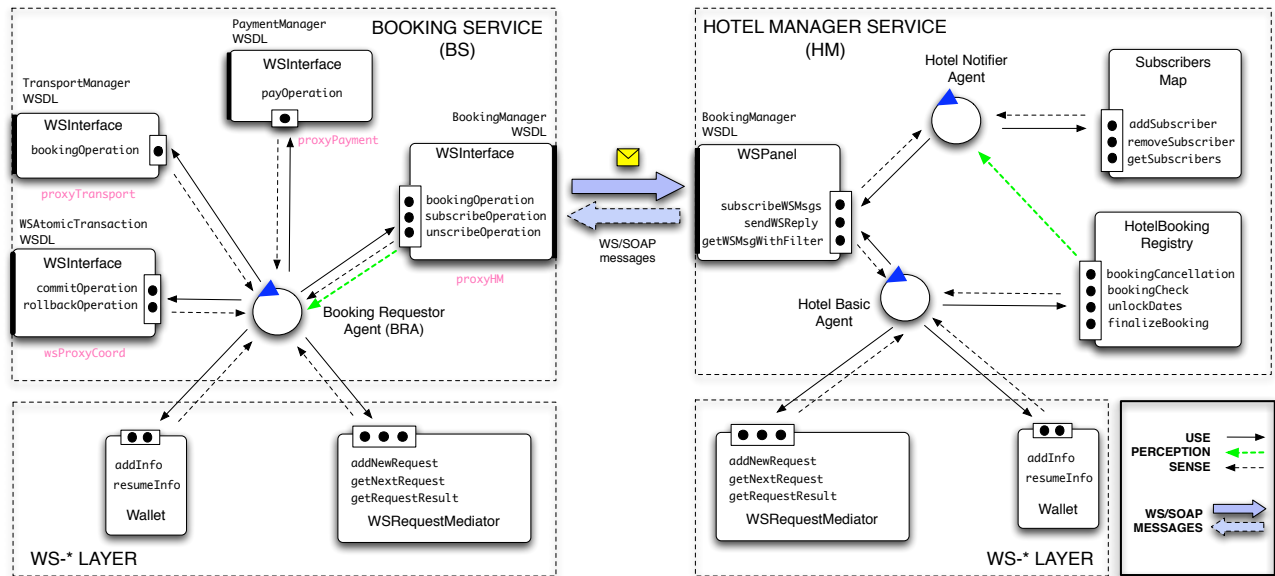


Fig. 2. Structural architecture showing the services involved in the *Book an Holiday* scenario. On the left side, the *Booking Service* is controlled by a Booking Requestor Agent managing WSInterface artifacts wrapping services as *Transport Manager*, *Payment Manager*, *Hotel Manager* and *WSAtomicTransaction*. On the right side, the *Hotel Manager Service* uses two agents (*Hotel Notifier* and *Hotel Basic*) and two artifacts (*Subscribers Map* and *HotelBooking Registry*) in order to provide the booking service and the notification events exploitable by the users. The two services make use of an additional layer (on the bottom in figure) in which specialized agents and artifacts coordinate the transactions according to WS-* protocols.

As showed in Fig. 2, the application is centered on two main services: *Booking Service* and *Hotel Manager*. The *Hotel Manager* (HM) service manages the booking tasks and also provides notification functionalities to subscribers. HM has been designed using two specialized agents, the *Hotel Basic Agent* and *Hotel Notifier Agent*, sharing and exploiting an instance of *WSPanel* to expose the service (see Fig. 2 right).

To support their tasks, the agents providing the HM service use additional artifacts. In particular, in order to manage the requests related to bookings and cancellations *Hotel Basic Agent* exploits the functionalities provided by an *HotelBookingRegistry* artifact. Besides, in order to manage the HM's notification services *Hotel Notifier Agent* uses a *SubscribersMap* artifact. It is assumed to keep track of the subscriptions requested and monitor the *HotelBookingRegistry* so as to notify interested subscribers as soon as changes regarding date availabilities are observed. Notice that *SubscribersMap* and *HotelBookingRegistry* represent the external resources needed by agents to achieve their goals in the context of this specific application (i.e. business artifacts)

On the user side, the *Booking Service* (BS) realizes the task related to a client agent who wants to organize an holiday. The service is built around the role played by a *Booking Requestor Agent* (BRA), whose final goal is to plan the required reservation related to an holiday for a given date. To achieve this goal, BRA is assumed to compose several resources, in this case related to the use of artifacts embedding external web services (see Fig. 2 left): In this case, the *Hotel Manager* service (HM) is used to (i) check the availability of hotel rooms for the specified period, (ii) subscribe for possible notifications (in case of missed availability) and (iii) finalize the reservation.

Besides HM, the *Booking Service* uses additional services to accomplish its goal. In particular, a *TransportManager* service (TM) is needed to manage the booking for the transports used for arriving to (and leaving from) the specified destination. A *PaymentManager* service (PM) is used to manage bank accounts and to finalize the payment. As showed in Fig. 2, in order to externalize the computational load required to manage complex messaging, the *Booking Requestor Agent* in this case exploits the support provided by the WS-* layer (i.e., *Wallet* and *WSRequestMediator* artifacts). In addition, BRA main task is further managed through an atomic transaction (WS-AT) involving the overall set of services realizing the *booking application*. A dedicated proxy is then used to involve an external coordination service.

B. Agents and Web Services Implementation

Part of the implementation of the *Book an Holiday* Scenario is here described through the specification, in *Jason*, of the BRA agent (TABLE II shows a relevant code fragment). Agent's specification is provided in a goal-oriented format, assuming different plans addressed to a precise step in the business task to achieve.

The initial goal for BRA is to initiate a a booking activity (+!start_booking, line 00 in TABLE II). In so doing, BRA launch a series of sub-level goals. An initial +!setupTools (line 05) is executed to retrieve or create the needed artifacts (which identifiers are stored as beliefs in the form *artifact_id(a_name, a_id)*). The WS-AT context for managing the booking is then retrieved from *RequestMediator* artifact residing in the WS-* layer, and then stored into the *Wallet* as an *ATContext* info (line 09). A *WSInterface* artifact is also created (line 10) for

interacting with the HM. Its artifact identifier is then stored as a belief (`artifact_id(proxyHM, ProxyID)`) and the context related to the ongoing WS-AT is used to configure it (line 13). For simplicity, a series of agent's sub-goals are here not fully specified and concerns low level computation performed for instance to manage data and to interact with additional resources. Among others, for instance, the plan `retrieveDate` (line 02) is executed to retrieve the information provided – for instance – by a human user, and to store it in form of agent's belief `date(Date)`.

As showed in Fig. 3, BRA's terminal goal is managed by a workflow of purposive activities, realized by specific plans, as they are specified by the `#!book_an_holiday` goal (line 16). The first activity consists in booking the hotel for the given dates (line 16): after having specified the context, thus retrieving the belief related to the `proxyHM`, a message for the WS request is prepared (line 24) and the HM is used by the mean of a request-response protocol (lines 25–26). We may assume that the hotel has already reached the maximum amount of reservations for (some of) the dates in the requested period (the information about date availability is stored in the HM service by the `HotelBookingRegistry`, that is an artifact implemented at the application level). In that case, the HM service replies to BS with a message notifying the inability to finalize the reservation: this message is then analyzed by a special `inspect_h_response` plan that can provide an `available` or `not_available` result. The returned literal is then matched to verify the success of the booking operation (lines 27). In so doing, a fail event will occur whether the booking operation has failed and the `Res_H` is `not_available` (line 28). Thanks to the `Jason` execution model, this fail event causes the root plan to fail too. Hence, the failure can be suddenly handled by a `#!book_an_holiday` plan (line 43 and Fig. 3), by which the agent can subscribe to the HM with the aim to be notified whether some new availability is signalled. So far, in the hope that some client will cancel a reservation for the desired date, the agent focuses the HM proxy (`WSInterface`) and uses it for subscribing itself for the notification of possibly further availability (lines 43–47), then waits for a possible HM's notification. In this case (line 61–65) a `#!finalize` plan is assumed to manage a rollback of the service transaction. The WS-AT is coordinated through a `Coordinator Service` which is installed in a programmable infrastructure (WS-* layer) together with the set of the services required by WS-Coordination specification.

Each BRA's subscription is handled within the HM service by the `Hotel Notifier Agent`, which stores the request in the `SubscribersMap` business artifact (the structural description of the HM service is in Fig. 2, right). If, in the meanwhile, some other agent interacting with the HM cancels its reservation for the subscribed date, such a change is signalled – within the HM side – to the `HotelBookingRegistry` artifact, which stores the data related to the various reservations. In this case, the `Hotel Notifier Agent` is supposed to receive a percept from the registry: as soon as a `+data_status_changed` signal is perceived, the `Hotel Notifier Agent` creates a new sub-

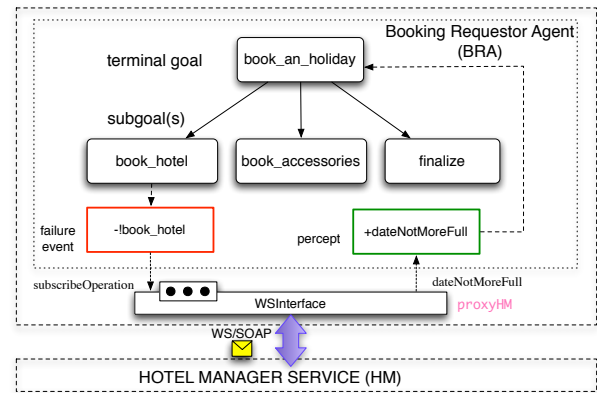


Fig. 3. Goal Decomposition Tree for Booking Requestor Agent (BRA) shows the structure of the various plans related to each sub-activity needed to achieve the terminal goal. Notice the interaction with `proxyHM` artifacts, in particular for the subscribe operation, performed after a failure in the `book_hotel` plan, and the execution of a new `book_an_holiday` plan, once a new availability is signalled by the HM.

goal to process such information, by retrieving the subscribers matching the given date, and by sending back a notification message to the BS who subscribed. Once a new availability occurs, the message coming from the HM arrives to the BS, and it is automatically translated by `WSInterface` and then it is signalled to the BRA agent. Also in this case, the event is received in form of percept and it succeeds to awaken the focusing BRA: the arriving percept `+dateNotMoreFull(Dates) [source(proxyHM)]` contains a date identifier (`Dates`) by which the agent can match the event and thus recognize it as a meaningful one, with respect to its goals (lines 36–37). In so doing, the BRA can now adopt a new instance of the `book_an_holiday` goal (line 39 and Fig. 3), by which the activities needed to achieve the goal are replanned from scratch. Differently from what happened in the first attempt, the BRA now finds the resources to succeed to book the hotel for the requested dates (HM response is, in this case, `available`).

Given this, BRA can now proceed with the following activity (`#!book_accessories`, line 30). It contacts the transport service and the payment manager, and, after having received the responses (line 33–38), it can control the results and, in so doing, achieve the terminal goal. Finally, the last activities `#!finalize(Res_H, Res_A)`, line 63) now commits the transaction upon the WS Coordination.

Some additional aspects are worth to emphasize in the described example. First, all mechanisms holding BRA to its idle state, during which it simply waits for a notification, as well as the mechanisms needed for its awaken, are here simply managed at a system level, both by `CARTAgO-WS` and `Jason` platforms. Once a message coming from the HM services arrives indicating an availability, the agent is suddenly and asynchronously awakened by the percept produced by the `WSInterface`. In so doing, the developer only needs to specify under which context the events coming from a given `WSPanel/WSInterface` artifact should be exploited to reactivate the agent practical reasoning. At the same time the programming model allows to automatically handle noticeable

events as failures (as in the case of missing availability in the booking task). In this case the programmer can suitably specify a purposive activity to recover the ongoing plans. The presented scenario also enlightens the support provided by the artifact based infrastructure for agent business activities. In this case the BRA only need to locate a `WSRequestMediator` and update a personal `Wallet` in order to automatically face the computational load related to the managing of the additional services needed by the protocol.

IV. CONCLUSION AND FUTURE WORKS

More and more agent technologies are recognized as a main actor in the engineering of service-oriented systems. Despite of this fact, few works have explored in literature the use of agent-oriented programming languages – and in particular those based on a strong notion of agency, such as BDI ones – to this end. In that perspective, we described a general-purpose programming model and platform for developing Web Services and SOA applications. The approach promotes the adoption of BDI agents programmed with proper agent languages/platforms (Jason is used in the paper) working together in artifact-based environments (constructed with `CArtAgO` technology). Agents work environments are instrumented, in particular, with artifacts specialized to provide functionalities useful for exploiting (and hiding) WS protocols and related technologies (`CArtAgO-WS` extension).

In conclusion, a couple of aspects are worth to emphasize. First, the programming model promotes a uniform approach to design complex service/application business logic in terms of structured goal-oriented activities. Indeed, agents' practical reasoning allows, for instance, to handle complex course of events and manage failures in a situated way, promoting coordination, adaptiveness, cooperation and so forth. Second, the use of an extensible artifact-based layer makes it possible to transparently manage the computational load required for agents to conform to WS-* protocols.

Besides improving the support to WS-* technologies, a major objective of future works will be the use of the platform to investigate the synergy between goal-oriented and artifact-based technologies for the construction of complex SOA/WS systems, with aspects concerning, for instance, goal-oriented orchestration [9], [26], goal-oriented business process management [5] and autonomic SOA/WS [12].

REFERENCES

- [1] M. Banzi, G. Caire, and D. Gotta. Wade: A software platform to develop mission critical. applications exploiting agents and workflows. In *AAMAS Industry Track*, 2008.
- [2] R. Bordini and J. Hübner. BDI agent programming in AgentSpeak using Jason. In F. Toni and P. Torroni, editors, *CLIMA VI*, volume 3900 of *LNAI*, pages 143–164. Springer, Mar. 2006.
- [3] R. Bordini, J. F. Hübner, and M. Wooldridge. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons, Ltd, 2007.
- [4] L. Bozzo, V. Mascardi, D. Ancona, and P. Busetta. COOWS: Adaptive BDI agents meet service-oriented computing (extended version). In *European Workshop on Multi-Agent Systems (EUMAS 2005)*, 2005.
- [5] B. Burmeister, M. Arnold, F. Copaciu, and G. Rimassa. BDI-Agents for Agile Goal-Oriented Business Processes. In *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008), Industry and Application Track.*, 2008.
- [6] F. Curbera, D. F. Ferguson, M. Nally, and M. L. Stockton. Toward a programming model for service-oriented computing. In *Third International Conference on Service-Oriented Computing (ICSOC-05)*, volume 3826 of *Lecture Notes in Computer Science*. Springer, 2005.
- [7] M. Dastani. 2APL: a Practical Agent Programming Language. *Autonomous Agents and Multi-Agent Systems*, 16(3):214–248, 2008.
- [8] R. T. Fielding and R. N. Taylor. Principled Design of the ModernWeb Architecture. *ACM Transactions on Internet Technology*, 2:115–150, 2002.
- [9] M. Georgeff. Service Orchestration: The Next Big Thing. *DM Review*, 2006.
- [10] D. Greenwood and M. Calisti. Engineering web service-agent integration. In *Proc. of IEEE Conf. on Systems, Man and Cybernetics*, 2004.
- [11] D. Greenwood, M. Lyell, A. Mallya, and H. Suguri. The IEEE FIPA approach to integrating software agents and web services. In *Proc. of Autonomous agents and multiagent systems (AAMAS-07)*, 2007.
- [12] S. A. Gurguis and A. Zeid. Towards autonomic web services: achieving self-healing using web services. *SIGSOFT Softw. Eng. Notes*, 30(4):1–5, 2005.
- [13] M. N. Huhns. A research agenda for agent-based Service-Oriented Architectures. In M. Klusch, M. Rovatsos, and T. Payne, editors, *CIA 2006*, volume 4149 of *LNAI*, pages 8–22. Springer-Verlag Berlin Heidelberg, 2006.
- [14] M. N. Huhns, M. P. Singh, and M. e. a. Burstein. Research directions for service-oriented multiagent systems. *IEEE Internet Computing*, 9(6):69–70, Nov. 2005.
- [15] X. T. Nguyen and R. Kowalczyk. WS2JADE: Integrating web service with jade agents. In *Service-Oriented Computing: Agents, Semantics, and Engineering*, vol. 4507 *LNCS*. Springer, 2007.
- [16] A. Omicini, A. Ricci, and M. Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17 (3), Dec. 2008.
- [17] M. Piunti, A. Ricci, L. Braubach, and A. Pokahr. Goal-Directed Interactions in Artifact-Based MAS: Jadex Agents Playing in `CArtAgO` Environments. In *Proc. of Web Intelligence and Intelligent Agent Technology (WI-IAT '08)*, Sydney, 2008. IEEE/WIC/ACM.
- [18] A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: A BDI reasoning engine. In R. Bordini, M. Dastani, J. Dix, and A. E. F. Seghrouchni, editors, *Multi-Agent Programming*. Kluwer, 2005.
- [19] A. Ricci and E. Denti. simpA-WS: A Simple Agent-Oriented Programming Model & Technology for Developing SOA & Web Services. In *Proceedings of AI*IATABOO Joint Workshop From objects to Agents (WOA 2007)*, 2007.
- [20] A. Ricci, E. Denti, and M. Piunti. A Platform for Developing SOA/WS Applications as Open and Heterogeneous Multi-Agent Systems. *Multi Agent and Gris Systems*, (to appear).
- [21] A. Ricci, M. Piunti, L. D. Acay, R. Bordini, J. Hübner, and M. Dastani. Integrating Artifact-Based Environments with Heterogeneous Agent-Programming Platforms. In *Proc. of the Seventh International Conference on Autonomous Agents and Multiagent Systems (AAMAS'08)*, pages 225–232, 2008.
- [22] A. Ricci, M. Piunti, M. Viroli, and A. Omicini. *Multi-Agent Programming: Languages, Tools and Applications. (Eds.) 2009*, Springer. ISBN: 978-0-387-89298-6, chapter Environment Programming in `CArtAgO`, pages 259–288. Springer, 2009.
- [23] A. Ricci and M. Viroli. simpA: An agent-oriented approach for prototyping concurrent applications on top of Java. In *Proc of Principles and Practice of Programming in Java (PPPJ-07)*, 2007.
- [24] G. Rimassa, M. E. Kermland, and R. Ghizzioli. Ls/abpm - an agent-powered suite for goal-oriented autonomic bpm. In *Demo Session in AAMAS 2008*, 2008.
- [25] A. A. Shafiq, H. F. Ahmad, and H. Suguri. AgentWeb Gateway - a middleware for dynamic integration of multi agent system and web services framework. In *IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*, 2005.
- [26] M. B. van Riemsdijk and M. Wirsing. Using goals for flexible service orchestration - a first step. In *Service-Oriented Computing: Agents, Semantics, and Engineering (SOCASE'07)*, vol. 4504 *LNCS*. Springer, 2007.

Exploiting Agents and Ontologies for Type- and Meaning-Safe Adaptation of Java Programs

Davide Ancona and Viviana Mascardi
 DISI, University of Genova,
 Via Dodecaneso 35, 16146, Genova, Italy
 {davide,mascardi}@disi.unige.it

Abstract—This paper discusses an application of intelligent software agents and ontologies to solve the problem of semi-automatic porting of Java programs.

We have designed a system for aiding users to adapt Java code in a type- and meaning-safe way, when an application has to migrate to new libraries which are not fully compatible with the legacy ones.

To achieve this, we propose an approach based on an integration of the two type-theoretic notions of subtyping and type isomorphism with ontology matching. While the former notions are needed to ensure flexible adaptation in the presence of type-safety, the latter supports the user to preserve the meaning of names that appear in the program to be adapted.

Intelligent agents control the different components of the system and interact with other agents in order to provide the final user with the semi-automatic porting service he/she required.

I. INTRODUCTION

Migrating a Java program p that uses library l into a corresponding program p' that uses library l' in a semi-automatic way is an open problem for which *no satisfying solution has been found yet*.

One aspect that must be considered while facing this problem, and that makes it hard to solve, is that migration must be type-safe. Replacing method m defined by l and used in program p by m' defined in l' , thus leading to a new program p' , is a legitimate operation only if no type inconsistencies are raised by this replacement. If the functionality of m and m' is the same no type problems will arise. But what should it happen in case of a difference in the type returned by m and m' , or in the type of some of their parameters, or in their number and order? The most conservative approach would be to give up, and to consider the migration possible only if elements of l used by p have corresponding elements in l' whose type is identical or isomorphic.

However, this is a very restrictive choice with little motivation: type identity or isomorphism between elements of l and the corresponding elements of l' may be relaxed by requiring that the type τ' of e' in l' is a subtype of the type τ of e in l , for a suitable definition of the subtype relation. This requirement allows a type-safe replacement of e in p with e' in p' .

For example, R. Di Cosmo, F. Pottier and D. Rémy propose an efficient decision algorithm for subtyping recursive types modulo associative commutative products that demonstrates the feasibility of using subtyping instead of type isomorphism, when translating a program into another [1].

The limitation of their work, that we want to overcome by exploiting intelligent agents and ontologies in our system, is that they abstract from the names of classes, methods and attributes and just consider safe matching between types. Since there may be a large number of type correspondences $\langle \tau, \tau' \rangle$ that preserve type-safety, re-introducing names of classes, methods and attributes into the algorithm that matches libraries' elements may help in removing those correspondences that, even if type safe, are not "meaning-safe". Correspondences between names of methods and attributes are also needed during the translation process where type correspondences are not enough.

Assume that we would like to port p from l to l' . For simplicity, the problem can be reduced to the following example scenario: p is the program

```
AttributeList atts;  
String name = atts.getName(0);
```

and l is defined as follows:

```
class AttributeList extends Object {  
  String getName(int i){...}  
}
```

where `Object` and `String` are the usual predefined classes defined in the standard package `java.lang`.

The library l' to which p has to be ported contains the following class declarations:

```
class Attributes extends Object {  
  int   getLength(){...}  
  String getLocalName(int index){...}  
  String getAttributeType(int index){...}  
}
```

The approach discussed in [1] would tell us that the structural types of `AttributeList` and `Attributes` are compliant because of a combination of isomorphism and subtyping. Or, in other words, would tell us that the correspondence $\langle \text{AttributeList}, \text{Attributes} \rangle$ is type safe. This is a useful information, but it does not help us in automatically translating p into p' in order to use l' .

What we would like to have, instead, is the set of correspondences $\{\langle \text{AttributeList}, \text{Attributes} \rangle, \langle \text{getName}, \text{getLocalName} \rangle\}$. This set cannot be obtained by just checking the type compliance of `String getName(int)`

with `int` `getLength()`, `String` `getLocalName(int)`, and `String` `getAttributeType(int)`.

In fact, while `getLength` is not type compliant with `getName`, both `getLocalName` and `getAttributeType` are. However, we expect that the right correspondence is that between `getName` and `getLocalName`, due to the intended meaning of their names.

It is here that **ontologies** come into play: assuming that an “ontology matching algorithm” can devise the correspondences between ontology elements (classes, properties, relationships, individuals) that better respect their intended meaning, and assuming that from a Java library, an ontology carrying the intended meaning of the library elements can be extracted, we propose to extract ontologies o and o' from l and l' , and to run a matching algorithm on them.

And it is here that **agents** come into play: the system that we have designed consists of complex components that must provide different kinds of services (type and ontology extraction, type and ontology matching, filtering of the matching results, assisted extraction of the translation function, actual translation) either to the final user or to other system’s components. In order to make our system as flexible as possible, we associate an intelligent agent with each component. The agent controls the component and interacts both with other agents and with the user.

The output of the type and ontology matching algorithms, controlled by a *Type Matching Agent* and by an *Ontology Matching Agent* respectively, will be combined by a *Filtering Agent* in order to produce a type- and meaning- safe matching relation. A human user assisted by a *Function Extraction Assistant Agent* will disambiguate multiple possible matchings in order to identify a *match* function which will finally be used by a *Translation Agent* to translate p into p' .

Continuing the example above, p' would be

```
Attributes atts;
String name = atts.getLocalName(0);
```

where `Attributes = match(AttributeList)` and `getLocalName = match(getName)`. Thanks to the *match* function, the translation from p to p' can be fully automatized.

The aim of this paper is to discuss a multiagent system that exploits type and ontology matching techniques to make automatic migration of Java programs possible. The paper is organized in the following way: Section II describes the architecture of our multiagent system and Sections III and IV describe the *Ontology Extraction* and *Ontology Matching* agents in detail. Section V concludes and highlights future directions of work.

II. ARCHITECTURE

The purpose of our multiagent system, depicted in Figure 1, is to provide the service of computing a *match* function between the elements of two Java libraries l , l' given in input either by a human user or by any other software application, by exploiting interactions among the different agents belonging

to it¹. If the user (agent, software application) wants the additional service of performing the translation of a Java program p that uses library l into a Java program p' that uses l' , the *match* function can in turn be given in input to the Translation Agent which computes a translation p' of p driven by *match*.

The *match* function is obtained in the following way: ontologies o and o' are extracted from libraries l and l' respectively. In a similar way, collections of types t and t' are extracted from l and l' .

The Ontology Matching Agent interacts with a set of Simple Ontology Matching agents (SOM_{*i*} in Figure 1), each in charge of running one specific ontology matching algorithm chosen from a pool of existing ones (see Section IV, last paragraphs). The Ontology Matching Agent may decide to demand the ontology matching service to the SOM agent that has the lowest workload, to the one that seems more suitable to correctly match ontologies o and o' according to quality of service criteria or efficiency needs, or to any other SOM agent according to some policy including running all the available ontology matching algorithms and either merging the obtained results or selecting one of them based on ex-post analysis². At the end, the Ontology Matching Agent obtains from one or more SOMs the alignments (namely, the sets of correspondences) a_1, a_2, \dots, a_n between o and o' and merges them or selects the most preferred alignment among them if it is the case. The Type Matching Agents behaves in the same way, controlling a set of Simple Type Matching agents (STM_{*j*} in Figure 1) each in charge of running a specific type matching algorithm on t and t' to get *tm*. The type match *tm* is used for selecting only those correspondences in a that are type safe. We name this activity “filtering”.

Filtering, whose responsibility is given to the Filtering Agent, still does not ensure that we obtain a set of correspondences that is a function: it might still be a relation, because more than one correspondence involving $e \in l$ is both type- and meaning-safe.

The user is involved in the loop for making the relation output by the Filtering Agent turn out into a *match* function: if many correspondences are possible for an element $e \in l$, the user will be asked to make his/her choice among them. Another information must be integrated into the *match* function, namely, for any method $m \in l$, which injection must be applied on its parameters p_1, \dots, p_n in order to obtain

¹Currently, some agents belonging to the MAS such as type matching and filtering agents have little decisional power and autonomy, so they could be collected into a single sequential process, simplifying the system design. However, we expect that these agents may be equipped with a higher degree of intelligence in a future version of the system. Hence, we model them as agents even if, in the current version, they are just service providers.

²Alignments can be compared according to their precision and recall. Unfortunately, computing precision and recall of an alignment between o and o' is only possible if a *reference alignment* for o and o' has already been developed by hand. In fact, precision is defined as the number of correctly found correspondences with respect to a reference alignment divided by the total number of found correspondences and recall is defined as the number of correctly found correspondences divided by the total number of expected correspondences. The higher the precision and recall, the better. If no reference alignment exists, only quantitative features of the alignment such as dimension, number of correspondences with the same first element, etc, can be considered to decide whether one alignment is “better” than another one.

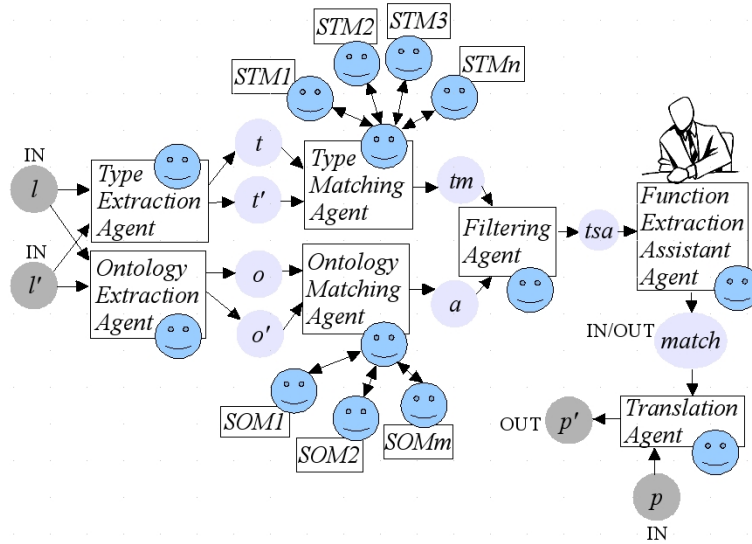


Fig. 1. The architecture of our multiagent system.

the tuple p_1, \dots, p_k , $k \leq n$ whose ordered elements can be used as parameters for $m' \in l'$, where $m' = match(m)$. Also in this case, the user may be required to make a choice if more injections are possible. For example method $m_1(c_1, \mathbf{int}, \text{String})$ in l might be type- and meaning-safely replaced by $m_2(\mathbf{int}, \text{String}, c_1)$ in l , but a permutation of its parameters is required when actually translating p that uses m into p' that uses m' .

The *match* function (which is indeed a family of functions working either on elements of l , or on tuples of elements of l) is needed by the Translation Agent.

Of course, it might also happen that the Filtering Agent cannot achieve its goal because there are some elements in l for which no corresponding element in l' has been found and thus no *match* function from l to l' can be computed. The user will be involved in this case too: the Filtering Agent will inform him/her that no type and meaning-safe matching was possible for some elements, and the result of the filtering stage will be shown to him/her. Even if no automatic translation of p will be possible due to the impossibility to generate a *match* function, the user might find the result of the Filtering Agent useful for driving his/her hand-made translation.

If, thanks to the human intervention, a *match* function has been defined, the automatic translation of p into p' can be performed by the Translation Agent, leading to the desired output, namely program p' .

In the sequel of this section, each agent is shortly presented. Agents that deal with ontologies are discussed in more detail in the next sections.

Ontology Extraction Agent

The Ontology Extraction Agent takes one Java library as input and returns an ontology that models the structure of the library in term of its classes, their subclass relationships, their methods and attributes. This agent, described in Section III, must operate on both l and l' in order to obtain o and o' respectively.

Type Extraction Agent

The Type Extraction Agent takes one Java library as input and returns a collection of types following S. Jha, J. Palsberg and T. Zhao's proposal [2], [3]. Since Java classes belonging to a library may mutually refer to one another, types in the collection may be mutually recursive. In our system, the Type Extraction Agent must operate on both l and l' in order to extract the corresponding collections of types, t and t' respectively.

Ontology Matching Agent

The service offered by the Ontology Matching Agents is returning an alignment of the two ontologies taken in input. This agent is responsible for the "meaning-safety" of the matching between elements of l and elements of l' ; it will take the ontologies o and o' extracted from l and l' respectively as input and will return an ontology alignment a between them. As we will discuss in Section IV, many ontology matching algorithms and tools exist: we will integrate the most relevant ones into our system by implementing, for each of them, a SOM agent that provides an interface towards the algorithm/tool. The Ontology Matching Agent will coordinate the activity of SOM agents

Type Matching Agent

Once the collections of types induced by l and l' have been extracted, a type-safe matching between them must be computed. The algorithm we will use for this activity is inspired by that proposed by R. Di Cosmo, F. Pottier and D. Rémy in [1] and is briefly described in [4]. It ensures the type-safety of the matching.

Filtering Agent

In order to find a matching between the elements of l and those of l' that is both type-safe and that takes the meaning

of names of methods, attributes and classes into account, as well as their structural relationships, we need to filter elements of a by taking the type-safe correspondences contained in tm into account. A Filtering Agent that implements the algorithms described in [4] has been designed to this aim.

Function Extraction Assistant Agent

In the general case the output of the Filtering Agent, tsa (for *type safe alignment*), will not be deterministic enough to be used for translating a program p that uses l into the corresponding program p' that uses l' . There might be elements of l that can be matched to more than one element in l' taking both types and meaning into account, and no algorithm could automatically determine the right choice. Once most of the work has been done and the subset tsa of $elements(l) \times elements(l')$ has been generated, the Function Extraction Assistant Agent comes into play and interacts with the user in order to complete the definition of the *match* function that will drive the translation from p to p' . The task of the user mainly consists in making choices among a set of possibilities provided by the Filtering Agent, in order to constrain a relation to become a function. The user is also asked to define the right operations to be performed on parameters of $m \in elements(l)$ in order to obtain a tuple of parameters suitable for the corresponding method $m' \in elements(l')$.

Of course there might be elements of l for which no type safe matching into a corresponding element of l' exist, and this would mean that tsa could never become a function, and that the system has nothing left to do. The user can benefit from knowing tsa , but he/she has to perform the translation from p to p' by hand.

Translation Agent

In case a the *match* function has successfully been extracted, the Translator Agent can provide its translation service by taking a function *match* and a program p and returning a program p' following the rules defined in Section 7 of [4]. The program p to migrate is given in input only to the Translation Agent. The matching function *match* only depends on l and l' : it can be reused for any p developed for using l which must be updated for using l' . The alternative of considering p from the earliest phases of the process has been taken into consideration because of some advantages it would give. In fact, knowing p since the beginning would allow the multiagent system to limit the extraction and matching activities only to those elements of the library that are actually used by p , as well as those that have some dependency relation with them. This would restrict the search space, but would also cause a loss of generality of the function *match*, which should become a $match_p$ function depending on p and might be used only for translating p and programs that use less elements of l than p . A program p_2 that uses only one more element from l w.r.t p would require the generation of a new $match_{p_2}$ function.

III. ONTOLOGY EXTRACTION AGENT

This section describes the algorithm for automatically extracting an OWL ontology from a Java library exploited by the

Ontology Extraction Agent. In case more ontology extraction algorithms should be implemented, the Ontology Extraction Agent might coordinate interface agents towards all or some of them, in the same way as the Ontology Matching and Type Matching agents do.

In order to explain how the extraction algorithm works, we need to provide some details on the subset of OWL that we will use for representing ontologies corresponding to Java libraries. We have designed the extraction in order to make this subset as small as possible. In particular, it is a proper subset of OWL Lite.

a) *Data Types*: Data Types used in OWL ontologies are those defined by the XML Schema specification, <http://www.w3.org/TR/xmlschema-2/>:

- *decimal* represents the subset of the real numbers, which can be represented by decimal numerals; *integer* is derived from decimal by fixing the number of decimal digits to 0, and disallowing the trailing decimal point. This results in the standard mathematical concept of the integer numbers. Neither decimal nor integer have a direct counterpart in Java primitive data types.
- *long* is derived from integer by setting the maximum value to be 9,223,372,036, 854,775,807 and the minimum one to be -9,223,372,036,854,775,808 (both included); it corresponds to the *long* Java primitive data type.
- *int* is derived from long by setting the maximum value to be 2,147,483,647 and the minimum value to be -2,147,483,648 (both included); it corresponds to the *int* Java primitive data type.
- *short* is derived from int by setting the minimum admissible value to -32,768 and the maximum admissible value to 32,767 (both included); it corresponds to the *short* Java primitive data type.
- *byte* is a short ranging between -128 and 127 (both included); it corresponds to the *byte* Java primitive data type.
- *float* is patterned after the IEEE single-precision 32-bit floating point type; it corresponds to the *float* Java primitive data type.
- *double* is patterned after the IEEE double-precision 64-bit floating point type ; it corresponds to the *double* Java primitive data type.
- *boolean* has the value space required to support the mathematical concept of binary-valued logic: {true, false}; it corresponds to the *boolean* Java primitive data type.

OWL primitive data types do not include *char*, which is the only Java primitive data type with no direct correspondence. However, since *char* is a finite-valued type type, it may be easily represented as an OWL class with a finite number of instances, as a set of integers with a maximum cardinality (the *owl:maxCardinality* built-in OWL property may be used to this aim), or in other straightforward ways. Instead, OWL primitive data types include for example *string*, *date*, *time* that correspond to some extent to the *String*, *Date*, *Time* classes provided by `java.lang` and `java.sql` packages, respectively.

Since OWL provides no data type corresponding to *void*, we assume that an OWL class named *Void* is defined in a names-

pace that we abbreviate with *myns*, and that it corresponds to the *void* type specifier in Java.

b) Namespace: Namespaces are inherited by OWL from XML. XML namespaces provide a simple method for qualifying element and attribute names used in XML documents by associating them with namespaces identified by URI references. A standard initial component of an ontology includes a set of XML namespace declarations that provide a means to unambiguously interpret identifiers and make the rest of the ontology presentation much more readable.

c) Class: A class defines a group of individuals that belong together because they share some common properties. The OWL class element, identified by `owl:Class`, is a subclass of the RDFS class element, `rdfs:Class`. The rationale for having a separate OWL class construct lies in the restrictions on OWL DL (and thus also on OWL Lite), which imply that not all RDFS classes are legal OWL DL classes.

d) Subclass: Class hierarchies may be created by making one or more statements that a class is a subclass of another class. This can be achieved by using the `rdfs:subClassOf` element defined by RDFS.

e) Property: Properties have originally being defined in RDF and can be used to state relationships between individuals (object properties, `owl:ObjectProperty`) or from individuals to data values (data type properties, `owl:DatatypeProperty`). Both object and data type OWL properties are subclasses of the RDF class `rdf:Property`.

A. From a Java library to an OWL ontology

The algorithm that we describe in this section has been designed for working under the assumption that names of methods and attributes of the classes in a class library are all different. The absence of name clashes between classes is given for granted, since a class library cannot include two classes with the same name. Even under the assumption that different classes with no inheritance relation among them define different methods, a preprocessing stage must be performed on the library in order to deal with method overriding. In fact, we cannot prevent subclasses from overriding methods defined in superclasses, but this leads to a violation of our assumption on disjoint names of methods. We deal with this situation by just removing the overridden method from all the subclasses that override it. This gives us two advantages:

- 1) the assumption under which the algorithm works is respected;
- 2) we avoid that a method m defined by class c may be matched to m' , and the same method m overridden by a subclass of c is matched to $m'' \neq m'$.

The basic ideas underlying the extraction algorithm are:

- The Java library l corresponds to a single OWL ontology lo named after the library name and defined in a namespace $l.ns$.
- Java classes belonging to l correspond to OWL classes belonging to lo ; the identifier of the OWL class coincides with the name of the Java class it corresponds to.

- If the Java class sc extends c , then the OWL class corresponding to c (that we name $owl(c)$ for our convenience) is defined as a subclass of the OWL class corresponding to sc .
- Since properties of an OWL class are inherited by its subclasses, the Java methods and attributes of class c are translated into OWL properties with identifier identical to their name and domain $owl(c)$. This allows them to be inherited by $owl(c)$ ' subclasses for free. The range of a property corresponding to a Java attribute is defined as the attribute's type; that of a property corresponding to a method is a pre-defined OWL class named `myns:MethodF`.

Our assumption of absence of clash names is very strong, but it allows us to describe the basic ideas underlying the algorithm in a clear and understandable way, discarding the technical details raised by name clashes. The reason for this assumption is that we translate all the elements (classes, attributes, methods) of the class library into corresponding elements of a unique OWL ontology. Unfortunately, an OWL ontology cannot include properties with the same name, even if their domain and range are different as it should happen with methods, parameters and attributes with the same name but different functionality.

In the real case, where name clashes between methods, parameters, and attributes may occur, two solutions have been devised.

- 1) Instead of translating the entire Java library into an OWL ontology, each Java class c should be translated into an OWL ontology o defined within a namespace ns created starting from c in a way that ensures its uniqueness. Methods and attributes of class c , as well as the methods' parameters, should be translated into properties of the ontology o within the namespace ns . The usage of different ontologies defined in different namespaces should allow us to identify each element of a Java class in a unique way, and thus to overcome the problem of name clashes (using the same identifier in different namespaces is, of course, admitted). The ontology corresponding to the Java class c should import all the ontologies corresponding to translations of Java classes referenced in c , and thus a pre-processing phase should be added to the extraction algorithm. The Java library l should be translated into an ontology that just imports all the ontologies corresponding to the Java classes belonging to l .

The main drawback of this approach, besides a much more complex extraction algorithm, is that few implemented matching algorithms that the Simple Ontology Matching Agents, SOMs, should interface take namespaces correctly into account.

- 2) The Java library should still be translated into a single OWL ontology, but clashing names should be modified during their translation in order to obtain an ontology "clash-free".

Here, the drawback is that the modification of names would result into poorer performances of the ontology

matching algorithms. If, for example, method m in the library l has been translated into $m14$ in ontology o because of a name clash, and method m in library l' has been translated into $m37$ in ontology o' , again because of a name clash, the confidence in the correspondence $\langle m \in o, m \in o' \rangle$ would turn out to be lower than the confidence in the correspondence $\langle m14 \in o, m37 \in o' \rangle$ for most matching algorithms, because of the syntactic difference between the two names.

The following paragraphs describe the extraction of the OWL elements starting from the Java library elements and provide examples.

OWL elements corresponding to Java classes

A Java class c that extends no class corresponds to an OWL class c (Table I).

A Java class sc that extends a class c different from Object corresponds to an OWL class sc defined as a subclass of c (Table II).

OWL elements corresponding to attributes of Java classes

An attribute a of class c whose type is a basic type t with a corresponding data type in XML corresponds to an OWL datatype property whose ID is a , whose domain is c , and whose range is the XML data type that corresponds to t (Table III).

An attribute a of class c whose type is the class c' defined in the Java library corresponds to an OWL object property whose ID is a , whose domain is c , and whose range is c' (Table IV).

OWL elements corresponding to methods of Java classes

Since we are not interested in representing the functionality of a method m in the ontology, we treat methods in the same way as attributes with the only difference that their range is always an OWL class defined in our namespace, and named "mysns:MethodF". The domain of a method is the OWL class representing the Java class it belongs to (Table V).

IV. ONTOLOGY MATCHING AGENT

The Ontology Matching Agent will coordinate Simple Ontology Matching Agents, each interfacing towards some existing algorithm and/or tool (for example those mentioned at the end of this section, but others might be considered).

In the recent past, the second author of this paper together with other colleagues from the University of Genova designed, implemented and tested a FIPA compliant Ontology Agent for JADE [5] that provides Ontology Matching services to a MAS [6]. We plan to extend such an agent by adding intelligence to it in the choice of the right matching algorithm (among existing ones) to use, based either on work-balance issues or on quality of service provided, or on both. The experiments described in [7] demonstrate that better results are achieved by more time-consuming algorithms. According to the user's needs, a faster algorithm might be preferred to a slower one, even if this might cause a degradation of the results' quality.

The Ontology Matching Agent will take the user's preferences into account for delivering the best service to each user.

In this section, we shortly review the state of the art of ontology matching systems and algorithms towards which Simple Ontology Matching Agent will interface. We draw inspiration from [8]. Following the terminology proposed there, a correspondence between an entity e belonging to ontology o and an entity e' belonging to ontology o' is a 5-tuple $\langle id, e, e', R, conf \rangle$ where:

- id is a unique identifier of the correspondence;
- e and e' are the entities (e.g. properties, classes, individuals) of o and o' respectively;
- R is a relation such as "equivalence", "more general", "disjointness", "overlapping", holding between the entities e and e' .
- $conf$ is a confidence measure (typically in the $[0, 1]$ range) holding for the correspondence between the entities e and e' ;

An alignment of ontologies o and o' is a set of correspondences between entities of o and o' , and a matching process is a function f which takes two ontologies o and o' , a set of parameters p and a set of oracles and resources r , and returns an alignment A between o and o' .

Two of the dimensions according to which matching techniques can be classified are the level (element vs structure) and the way input information is interpreted (syntactic vs external vs semantic).

Level: element vs structure

Element-level matching techniques compute alignments by analyzing entities in isolation, ignoring their relations with other entities. Structure-level techniques compute alignments by analyzing how entities appear together in a structure.

Element-level techniques include, among others:

- *String-based techniques*, that measure the similarity of two entities just looking at the strings (seen as mere sequences of characters) that label them. They include substring distance, Jaro measure [9], n -gram distance [10], Levenshtein distance [11], SMOA measure [12].
- *Language-based techniques*, that consider entity names as words in some natural language and exploit Natural Language Processing techniques to measure their similarity.
- *Constraint-based techniques*, that deal with the internal constraints being applied to the definitions of entities, such as types, cardinality of attributes, and keys.

Structure-level techniques include:

- *Graph-based techniques* that take the input ontology as a labeled graph.
- *Taxonomy-based techniques*, that are also graph algorithms which consider only the specialization relation.
- *Model-based techniques* that handle the input based on its semantic interpretation (e.g., model-theoretic semantics). Examples are propositional satisfiability (SAT) and description logics (DL) reasoning techniques.

public class Bike	<owl:Class rdf:ID="Bike"/>
-------------------	----------------------------

TABLE I
JAVA CLASS *c* THAT EXTENDS NO CLASS.

public class MountainBike extends Bike	<owl:Class rdf:ID="MountainBike"> <rdfs:subClassOf rdf:resource="Bike"/> </owl:Class>
---	---

TABLE II
JAVA CLASS *sc* THAT EXTENDS CLASS *c*.

Attribute cadence of the class Bike: public int cadence;	<owl:DatatypeProperty rdf:ID="cadence"> <rdfs:domain rdf:resource="Bike"/> <rdfs:range rdf:resource="xsd:int"/> </owl:DatatypeProperty>
---	--

TABLE III
ATTRIBUTE WITH A BASIC TYPE.

Attribute ft of the class Bike: public BikeFeatr ft;	<owl:ObjectProperty rdf:ID="ft"> <rdfs:domain rdf:resource="Bike"/> <rdfs:range rdf:resource="BikeFeatr"/> </owl:ObjectProperty>
---	---

TABLE IV
ATTRIBUTE WITH TYPE *c*.

Interpretation of input information: syntactic vs external vs semantic

Syntactic techniques interpret the input in function of its sole structure following some clearly stated algorithm.

External techniques exploit auxiliary (external) resources of a domain and common knowledge in order to interpret the input.

Semantic techniques use some formal semantics (e.g., model-theoretic semantics) to interpret the input and justify their results. In case of a semantic based matching system, a further distinction between exact algorithms (that guarantee a discovery of all the possible correspondences) and approximate algorithms (that tend to be incomplete) may be done.

Implemented matching systems and infrastructures

Many implemented matching systems and algorithms exist. If we just consider those listed in the “Project” section of the Ontology Matching portal, <http://www.ontologymatching.org/projects.html>, we may count about thirty of them. These systems and infrastructures are very different one from another. Many of them have been carefully analyzed and compared in [8], as well as in previous works by the same authors [13], [14] and by other researchers [15].

Just to cite some very recent systems, HMatch [16], [17] is an automated ontology matching system able to handle

ontologies specified in OWL. Given two concepts, HMatch calculates a semantic affinity value as the linear combination of a linguistic affinity value and a contextual affinity value. For the linguistic affinity evaluation, HMatch relies on a thesaurus of terms and terminological relationships automatically extracted from the WordNet lexical system. The contextual affinity function of HMatch provides a measure of similarity by taking into account the contextual features of the ontology concepts.

CtxMatch [18], [19] is a sequential system that translates the ontology matching problem into the logical validity problem and computes logical relations, such as equivalence, subsumption between concepts and properties.

The Alignment API [20] is an API and implementation for expressing and sharing ontology alignments. It operates on ontologies implemented in OWL and uses an RDF-based format for expressing alignments in a uniform way. The Alignment API offers services for storing, finding, and sharing alignments; piping alignment algorithms; manipulating (thresholding and hardening); generating processing output (transformations, axioms, rules); comparing alignments. The last release, Version 3.5, dates back to October, 21th, 2008.

AUTOMS-F [21] is a framework implemented as a Java API which aims to facilitate the rapid development of tools for automatic mapping of ontologies by synthesizing several

<pre> Methods setFeatr and getFeatr of the class Bike: public void setFeatr (BikeFeatr newFeatr, String newOwnerName, int newOwnersNum) { ... } public BikeFeatr getFeatr() { ... } </pre>	<pre> <owl:ObjectProperty rdf:ID="setFeatr"> <rdfs:domain rdf:resource="Bike"/> <rdfs:range rdf:resource="myns:MethodF"/> </owl:ObjectProperty> <owl:ObjectProperty rdf:ID="getFeatr"> <rdfs:domain rdf:resource="Bike" /> <rdfs:range rdf:resource="myns:MethodF"/> </owl:ObjectProperty> </pre>
--	--

TABLE V
METHODS.

individual ontology mapping methods. Towards this goal, AUTOMS-F provides a highly extensible and customizable application programming interface. AUTOMS [22] is a case study ontology mapping tool that has been implemented using the AUTOMS-F framework.

Finally, automatic matching techniques that exploit “Upper Ontologies”, namely general ontologies that deal with concepts that are the same across different domains, have been implemented and analyzed in [7].

V. CONCLUSION AND FUTURE WORK

In this paper we have described a multiagent system that, once implemented, should allow a user to semi-automatically porting a Java program p that uses library l to a program p' that uses l' in a type-safe and “meaning-safe” way. To the best of our knowledge, no previous attempts of exploiting agents and ontologies for facing porting and migration problems exist. We devise some similarity between our proposal and the Natural Programming Project, <http://www.cs.cmu.edu/~NatProg/>, working on making programming languages and environments easier to learn, more effective, and less error prone. The report [23] suggests that AI tools such as agents, advice, and reversible debuggers may help users convert their intentions into precise programs. In this paper we do not face the general problem of supporting the user in his/her programming activities: we face the more specific problem of helping the user in a migration problem with respect to the Java language. Nevertheless, our exploitation of intelligent agents for supporting the user in activities related to smart programming is coherent with the purpose of the Natural Programming Project.

The contribution of this paper is twofold. On the one hand, we have designed the multiagent system’s architecture; on the other hand, we have either identified existing algorithms to integrate in the agents when possible, or designed new ones (the ontology extraction algorithm described in this paper and the algorithms implemented by the Filtering and the Translation agents described in [4] are all original contributions).

The first activity we will carry out in the very near future is the implementation of the algorithms that, at this stage, are only designed. In parallel to the implementation of these

algorithms, the choice of the most suitable algorithms and tools to be accessed by Simple Ontology Matching Agents will be made.

Once all these components will be available and tests will be performed over them, a prototype demonstrating the feasibility of our approach will be created in JADE.

ACKNOWLEDGEMENTS

The authors acknowledge the anonymous reviewers for their thoughtful and constructive suggestions.

This work has been partially supported by MIUR EOS DUE - Extensible Object Systems for Dynamic and Unpredictable Environments, and by the CINI-FINMECCANICA Iniziativa Software project.

REFERENCES

- [1] R. D. Cosmo, F. Pottier, and D. Rémy, “Subtyping recursive types modulo associative commutative products,” in *TLCA 2005, Proceedings*, ser. LNCS, P. Urzyczyn, Ed., vol. 3461. Springer, 2005, pp. 179–193.
- [2] J. Palsberg and T. Zhao, “Efficient and flexible matching of recursive types,” in *LICS 2000, Proceedings*. IEEE Computer Society, 2000, pp. 388–398.
- [3] S. Jha, J. Palsberg, and T. Zhao, “Efficient type matching,” in *FOSSACS 2002, co-located with ETAPS 2002, Proceedings*, ser. LNCS, M. Nielsen and U. Engberg, Eds., vol. 2303. Springer, 2002, pp. 187–204.
- [4] D. Ancona and V. Mascardi, “Ontology matching for semi-automatic and type-safe adaptation of Java programs,” DISI - University of Genova, Tech. Rep., 2008, <ftp://ftp.disi.unige.it/person/AnconaD/AM1208.pdf>.
- [5] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*. Wiley, 2007.
- [6] D. Briola, A. Locoro, and V. Mascardi, “Ontology agents in FIPA-compliant platforms: a survey and a new proposal,” in *WOA'08, Proceedings*, M. Baldoni, M. Cossentino, F. D. Paoli, and V. Seidita, Eds. Seneca Edizioni, 2008.
- [7] V. Mascardi, A. Locoro, and P. Rosso, “Automatic ontology matching via upper ontologies: A systematic evaluation,” 2009, *IEEE Trans. Knowl. Data Eng.*, to appear.
- [8] J. Euzenat and P. Shvaiko, *Ontology Matching*. Springer, 2007.
- [9] M. Jaro, “UNIMATCH: A record linkage system: User’s manual,” U.S. Bureau of the Census, Washington (DC US), Tech. Rep., 1976.
- [10] E. Brill, S. Dumais, and M. Banko, “An analysis of the askmsr question-answering system,” in *EMNLP 2002, Proceedings*, 2002.
- [11] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals,” *Doklady akademii nauk SSSR*, vol. 163, no. 4, pp. 845–848, 1965, in Russian. English Translation in *Soviet Physics Doklady* 10(8), 707-710, 1966.

- [12] G. Stoilos, G. B. Stamou, and S. D. Kollias, "A string metric for ontology alignment," in *ISWC 2005, Proceedings*, ser. LNCS, Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, Eds., vol. 3729. Springer, 2005, pp. 624–637.
- [13] P. Shvaiko and J. Euzenat, "A survey of schema-based matching approaches," *J. Data Semantics IV*, vol. 3730, pp. 146–171, 2005.
- [14] P. Shvaiko, "Iterative schema-based semantic matching," DIT - University of Trento, Tech. Rep. DIT-06-102, 2006, ph.D. Thesis.
- [15] N. Choi, I.-Y. Song, and H. Han, "A survey on ontology mapping," *SIGMOD Record*, vol. 35, no. 3, pp. 34–41, 2006.
- [16] S. Castano, A. Ferrara, and S. Montanelli, "Matching ontologies in open networked systems: Techniques and applications," *J. Data Semantics V*, pp. 25–63, 2006.
- [17] S. Castano, A. Ferrara, and G. Messa, "ISLab HMatch Results for OAEI 2006," in *OM-2006, co-located with ISWC-2006, Proceedings*, 2006.
- [18] P. Bouquet, B. Magnini, L. Serafini, and S. Zanobini, "A SAT-based algorithm for context matching," in *CONTEXT 2003, Proceedings*, ser. LNCS, P. Blackburn, C. Ghidini, R. M. Turner, and F. Giunchiglia, Eds., vol. 2680. Springer, 2003, pp. 66–79.
- [19] P. Bouquet, L. Serafini, S. Zanobini, and S. Sceffer, "Bootstrapping semantics on the web: meaning elicitation from schemas," in *WWW 2006, Proceedings*, L. Carr, D. D. Roure, A. Iyengar, C. A. Goble, and M. Dahlin, Eds. ACM, 2006, pp. 505–512.
- [20] J. Euzenat and et al., "Alignment API and Alignment Server," 2008. [Online]. Available: <http://alignapi.gforge.inria.fr/>
- [21] A. Valarakos, V. Spiliopoulos, K. Kotis, and G. Vouros, "AUTOMS-F: A java framework for synthesizing ontology mapping methods," in *KOST '07, Proceedings*, 2007.
- [22] K. Kotis, A. G. Valarakos, and G. A. Vouros, "AUTOMS: Automated ontology mapping through synthesis of methods," in *OM-2006, co-located with ISWC-2006, Proceedings*, ser. CEUR Workshop Proceedings, P. Shvaiko, J. Euzenat, N. F. Noy, H. Stuckenschmidt, V. R. Benjamins, and M. Uschold, Eds., vol. 225. CEUR-WS.org, 2006.
- [23] H. Goodell, S. Kuhn, D. Maulsby, and C. Traynor, "End user programming/informal programming," *SIGCHI Bull.*, vol. 31, no. 4, pp. 17–21, 1999. [Online]. Available: <http://www.cs.uml.edu/~hgoodell/EndUser/blend/report.html>

The CAWE Framework - Enhancing Service Oriented Architecture with Context Awareness

Extended Abstract

L. Ardissono, R. Furnari, A. Goy, G. Petrone, M. Segnan
Dipartimento di Informatica
Università di Torino
Torino, Italy
Email: {liliana,furnari,goy,giovanna,marino}@di.unito.it

Abstract—The development of Web applications based on Service Oriented Architectures (SOA) is challenged by the lack of support to the specification of explicit context adaptation policies.

As an answer to this issue, we present the Context Aware Workflow Execution framework (CAWE), which enriches SOA with (a) context-aware workflow management; (b) dialog management capabilities supporting the adaptation of the interaction with the individual user, and (c) context-dependent User Interface generation.

I. INTRODUCTION

Context-awareness is particularly important in Web applications, which are accessed by large numbers of users, having diverse preferences, needs and capabilities, and using heterogeneous devices to interact with the business services. In order to suitably handle such variability, a self-managing system should be able to adapt both the service and the User Interface to the individual user and to the dynamic environment surrounding her/him.

However, Service Oriented Architecture (SOA, [1]), the reference model for the development of composite applications, does not explicitly deal with personalization and context-awareness. In fact, it embeds all the adaptation decisions in the process specifying the business logic of the applications.

In order to address this limitation, we designed a vertical SOA architecture which extends Service Oriented Computing with context-awareness and personalization capabilities. This extended abstract shortly presents the CAWE (Context Aware Workflow Execution) framework for the development of composite Web applications. The framework supports the adaptation of the business logic, interaction logic and User Interface to the users and to their context. Specifically, the framework supports:

- The context-dependent selection of the courses of action to be enacted, and of the service providers to be invoked, during the execution of the application.
- The generation of a context-dependent User Interface, tailored to the user's device (e.g., to its screen size) and to the user's preferences (e.g., background colors and font size).

- The management of tasks as dialogs with the user, the provision of extra-helpful information for non-expert users, and the management of a User Interface fitting the size of her/his device.

We exploited the CAWE framework to develop an e-Health prototype application supporting the management of a clinical guideline which coordinates the activities to be performed in order to monitor the health state of patients affected by heart diseases. The analysis of the e-Health domain, and the development of the application, proved the suitability of the adaptive features offered by our framework, as well as its applicability to real-world use cases.

II. THE CAWE FRAMEWORK

Service Oriented Architecture provides limited support to and context-awareness because it fails to recognize the central role of the adaptation logic and thus it embeds all the adaptation decisions in the workflow specifying the business logic of the applications. Specifically:

- As far as the business logic is concerned, the workflow underlying the applications embeds the variables to be taken into account and describes the alternative courses of action in a flat graph. Although this approach works well in simple cases, it does not scale to complex contexts.
- The User Interface (UI) and the interaction with the user lack flexibility because they are based on minimalistic techniques for the generation of device-dependent UI pages which fail to support the management of flexible dialogs with the user.

The CAWE framework supports the development of composite applications which tailor the business logic, the interaction with the user and the User Interface to the user and to her/his context. The key concept is the fact that the adaptation logic has to be explicitly represented. By extracting such logic from the application workflow, flexible techniques can be applied to steer the system behavior. The CAWE architecture includes two core components:

- The *Context Manager service (CtxMgr WS)* handles the context information during the execution of the application. Specifically, it handles a Role Model for each role

defined in the application workflow, as well as a User Model and a Context Model for each involved actor.

- The *Context-Aware Workflow Manager (CA-WF-Mgr)* enacts a context-sensitive workflow which defines the business logic of the application. For this purpose, it exploits two modules: the Workflow Adaptation Module shapes the workflow depending on the context; the workflow engine enacts the resulting workflow.

Within the CA-WF-Mgr, the *Dialog Manager* module acts as a bridge between the user and the workflow engine. When the user logs in the application, the Dialog Manager is invoked and takes the control of the interaction. The module adapts the User Interface to a context including both the user's device and her/his layout preferences.

A. business logic

In the CAWE framework, the business logic of an application is represented as a context-sensitive workflow organized in an abstraction hierarchy which specifies the system behavior at different levels of detail. Specifically:

- Besides the standard workflow activities (prescribing the invocation of service providers, the management of tasks, or some internal computation), a context-sensitive workflow can include *abstract activities* which describe a generic type of behavior, to be decided at runtime.
- Each *abstract activity* is associated with a set of *implementations* which describe different courses of action that the workflow engine should enact to complete the activity, depending on the context. Each implementation is a workflow which can specify rather different behaviors; e.g., starting a task to be performed by a human actor, invoking a Web Service, starting a complex subprocess, or carrying out some internal computation. Notice that an implementation may include itself some abstract activities; therefore, the context-sensitive workflow can be organized as a multi-level hierarchy.
- The *business logic adaptation policies* steer the selection of the implementations to be enacted during the execution of the abstract activities. These policies are described as (chains of) condition-action rules: the precondition of a rule is a boolean condition on context variables. The action can be the reference to another rule (rule chaining), or the name of the implementation to be enacted.

During the execution of the application, the workflow underlying the application is composed by recursively selecting the implementations of the abstract activities to be enacted, until the system's behavior is completely specified. This selection is steered by the business logic adaptation policies.

Specifically, the Context-Aware Workflow Manager wraps a workflow engine which executes the context-sensitive workflow as if it were a standard one. However, when the engine encounters an abstract activity, it invokes the Workflow Adaptation Module on the abstract activity. When the module returns the implementation to be enacted, the engine performs it as a subprocess of the main process instance. At subprocess

completion, the engine resumes the execution of the main workflow.

B. user interface

The Dialog Manager handles each task to be completed as a communicative goal to be achieved by carrying out a dialog with the user. Each dialog step is aimed at achieving a part of the task and is managed by generating a UI page. A Finite state Automaton specifies the interaction logic of the Dialog Manager: each state of the automaton corresponds to a UI page type and each state transition is performed as a consequence of a user action. The automaton describes the whole interaction with the user as far as task management is concerned. Specifically, a task is handled as follows:

- 1) The Dialog Manager sends the user's browser a personalized UI page representing an interaction turn. The page includes a set of input/output parameters to be acquired/presented and the navigation links enabling the user to continue the interaction. Moreover, the page includes a set of help links to get more specific information about the task and its parameters.
- 2) The user may perform different actions on the UI page: e.g., each help link, and each information link associated to the parameters, activate a nested dialog. Moreover, suitable transitions lead to the next, or to the previous step of the dialog, respectively.
- 3) At task completion time, the Dialog Manager notifies the workflow engine and feeds it with the acquired data.

The generation of the personalised pages is based on the evaluation of a set of UI adaptation policies, which steer the selection of the layout to be applied (given the user's preferences and device) and, consequently, determine the maximum number of parameters which can be put in each UI page.

Figure 1 shows a sample UI page, targeted to a desktop device, generated by our e-Health application during the management of a task (`storeTherapy()`).

- The top bar of the page includes the name of the application (eHealth) and reports the username (house) and the logout button.
- The middle bar of the page is organized as follows:
 - The higher portion shows the task name, the task ID (744) and the user's role (doctor).
 - The lower portion includes: a help link for the visualization of the task description; the position of the current interaction turn within the overall dialog (Page 1 of 2); the *continue* link (>>) taking to the next dialog turn, and the Cancel link.
- The lower part of the page is devoted to the visualization of the input and output parameters of the task. Specifically, the Form area shows the input ones, while the Information area displays the output ones. Each parameter name has a link to its more specific information (h).

Figure 2 shows the sequence of pages devoted to the same task, if the user uses a PDA to connect to the application. In order to cope with the smaller screen size, the dialog is performed in more steps than in the desktop case.



Fig. 1. First dialog turn in the management of task `storeTherapy`, tailored to a desktop device.



Fig. 2. First two dialog turns in the management of task `storeTherapy`, tailored to a PDA.

III. RELATED WORK

In Service Oriented Computing, some contributions extend standard Web Service composition languages with context-awareness features (e.g., C-BPEL; see [2]), in order to comply with Quality of Service (QoS) requirements. These approaches are affected by the limitations of standard Web Service composition languages, such as WS-BPEL ([3]) and its context-aware extensions (e.g., Context4BPEL, see [4]), which embed the adaptation logic in the workflow specification.

Our work overcomes the limitations of these works by introducing the abstract activities and by exploiting declarative

adaptation rules for the runtime, context-dependent selection of the courses of action to be enacted. In this way, the business logic of the application is shaped during its execution.

In the Semantic Web research, planning technology is applied to enhance the flexibility in Web Service composition. Moreover, plan-based approaches are applied to invoke Web Service providers in context-aware mode; e.g., see [5], [6], [7]. However, planning technology is not suitable to handle long-lasting services and processes because it does not support persistence management. Therefore, up to now it has only been used to handle short-lived composition plans. In fact, several proposals for the adoption of planners in Web Service

composition turned out to exploit workflow engines for the service execution; e.g., see [8] and [9].

Concerning the management of the interaction with the user, context-aware workflow systems only provide the adaptation of the User Interface (UI) to the user's device, in terms of stylesheet selection; e.g., see [7] and [10]. In comparison, CAWE supports applications which adapt both the code of the UI pages and the interaction logic to a complex context. Moreover, it supports the adaptation to multiple users, by tailoring the UI and the interaction logic on an individual basis.

In the research about dialog-based systems, some researchers employed scripts describing domain-level activities and linguistic behavior to model articulated task-oriented dialogs; e.g., see [11]. Moreover, planning technology was applied to manage short-lived interactions with the user; e.g., see [12]. We adopt Finite State Automata to handle the interaction with the user; although these are less flexible than plans, they are more robust and lightweight, and they support a predictable behavior.

IV. CONCLUSION

This extended abstract has presented the Context Aware Workflow Execution framework for the development of context-aware composite Web applications. The framework enriches Service Oriented Architecture with (a) adaptation techniques supporting the execution of context-sensitive workflows; (b) dialog management capabilities supporting flexible user interactions, and (c) context-dependent User Interface generation techniques aimed at presenting personalized information on different devices. As such, it supports the development of Web applications which can self-adapt to meet the requirements of heterogeneous users in dynamic usage environments.

More information about the CAWE framework can be found in [13], [14], [15].

REFERENCES

- [1] M. Papazoglou and D. Georgakopoulos, Eds., *Service-Oriented Computing*. Communications of the ACM, 2003, vol. 46, no. 10.
- [2] C. Ghedira and H. Mezni, "Through personalized web service composition specification: from bpel to c-bpel," *Electronic Notes in Theoretical Computer Science*, no. 146, pp. 117–132, 2006.
- [3] OASIS, "OASIS Web Services Business Process Execution Language," http://www.oasis-open.org/committees/documents.php?wg_abbrev=wsbpel, 2005.
- [4] M. Wieland, O. Kopp, D. Nicklas, and F. Leymann, "Towards context-aware workflows," in *Proc. Workshop on Ubiquitous Mobile Information and Collaboration Systems (UMICS 2007) at CAiSE'07*, Trondheim, Norway, 2007.
- [5] S. McIlraith, T. Son, and H. Zeng, "Semantic Web Services," *IEEE Intelligent Systems*, vol. 16, no. 2, pp. 46–53, 2001.
- [6] W. Balke and M. Wagner, "Through different eyes - assessing multiple conceptual views for querying Web Services," in *Proc. of 13th Int. World Wide Web Conference (WWW'2004)*, New York, 2004.
- [7] M. Keidl and A. Kemper, "Towards context-aware adaptable Web Services," in *Proc. of 13th Int. World Wide Web Conference (WWW'2004)*, New York, 2004, pp. 55–65.
- [8] D. J. Mandell and S. A. McIlraith, "Adapting BPEL4WS for the Semantic Web: The bottom-up approach to Web Service interoperation," in *LNCS 2870, Proc. 2nd International Semantic Web Conf. (ISWC 2003)*. Sanibel Island, Florida: Springer-Verlag, 2003, pp. 227–241.
- [9] G. Laures and K. Jank, "Adaptive Services Grid Deliverable D6.V-1. Reference architecture: requirements, current efforts and design," <http://asg-platform.org/cgi-bin/twiki/view/Public/ReferenceArchitecture>, Tech. Rep., 2005.
- [10] S. Ceri, F. Daniel, and M. Matera, "Extending webml for modeling multi-channel contextaware web applications," in *WISE - MMIS'03 IEEE Computer Society Workshop*, 2003.
- [11] J. Chu-Carroll and S. Carberry, "Collaborative response generation in planning dialogues," *Computational Linguistics*, vol. 24, no. 3, pp. 355–400, 1998.
- [12] C. Rich, D. McDonald, N. Lesh, and C. Sidner, "COLLAGEN: Java middleware for collaborative agents services with multiple suppliers," <http://www.merl.com/projects/collagen>, 2002.
- [13] L. Ardissono, R. Furnari, A. Goy, G. Petrone, and M. Segnan, "A framework for the management of context-aware workflow systems," in *Proc. of WEBIST 2007 - Third International Conference on Web Information Systems and Technologies*, Barcelona, Spain, 2007, pp. 80–87.
- [14] L. Ardissono, A. Goy, and G. Petrone, "A framework for the development of distributed, context-aware Adaptive Hypermedia applications," in *LNCS 5149, Adaptive Hypermedia and Adaptive Web-Based Systems, 5th Int. Conference, AH2008*, W. Nejdl, J. Kay, P. Pu, and E. Herder, Eds. Berlin Heidelberg New York: Springer-Verlag, 2008, pp. 259–262.
- [15] L. Ardissono, R. Furnari, A. Goy, G. Petrone, and M. Segnan, "A SOA-based model supporting adaptive web-based applications," in *Proc. of 3rd Conference on Internet and Web Applications and Services (ICIW 2008)*. Athens, Greece: IEEE, 2008, pp. 708–713.

AgentSeeker: an ontology-based enterprise search engine

Extended abstract

A. Passadore, A. Grosso, and A. Boccalatte, *University of Genova, Via Opera Pia 13, 16145 Genova*

Abstract— The aim of this paper is to introduce AgentSeeker: a distributed multi-agent platform for indexing local and online textual files, with the semantic contribution of domain specific ontologies. These ontologies describe the application domain and the competences the user is referring to, during the interaction with the platform, namely a query session. They are used by an Ontology Agent which organizes the results of a user's query, according to the concepts which represent the relevant entities in the company business.

AgentSeeker is addressed to enterprise applications, thanks to its flexible and scalable structure managed by the Federation Management Suite, which ensures a comfortable administration of the distributed platform, balancing the computational load.

Index Terms— Document, multi-agent system, ontology, search engine, MAS federation.

I. INTRODUCTION

Search engines represent a saving compass which enables us to find an Internet page winnowing the whole network or to find a personal document through a desktop application which parses private files.

Usual search engines denote an intuitive behaviour: they store the textual content of the parsed documents in a database and they return an ordered list of files containing the keywords suggested through a user's query. Proper algorithms calculate the rank for every hit and, according to this evaluation, the search engines display first the most relevant pages. In spite of this solution, it is a user's experience that sometimes the search engine gives a completely wrong page link due to a misunderstanding of the meaning of the keyword or neglects a page link which does not explicitly contain the given term but it is anyway relevant.

The aim of AgentSeeker, the search engine presented in this paper, is to make the document retrieval a more intelligent process, finding texts which are semantically bound to the user's query. In order to achieve this goal, two aspects of AgentSeeker are relevant: *software agents* and *ontologies*. Based on a multi-agent platform, AgentSeeker is a scalable and flexible solution which can be adapted to different contexts, thanks to the *AgentService Federation Management Suite*.

Even if AgentSeeker is not designed for competing with the giant search engines as *Google* or *Yahoo*, it is aimed both to index Internet pages and local files and it is especially focused to enterprise contexts where the value of the digital information is particularly high. A particular kind of agent is

able to manage ontologies, integrating the user's queries with semantically related words, discovered through the analysis of concept relations, specializations, and synonyms.

AgentSeeker is able to manage different amounts of textual documents: from a little corpus on a single file server, to a big collection scattered on a network. The agent roles involved in AgentSeeker are designed in order to operate in a variable amount of instances and to interact with peers in a circumscribed environment (a single PC) alike a distributed platform federation. And exactly the management of this distributed environment is a relevant topic of this extended abstract and it represents an improvement in respect with the first version of AgentSeeker presented in [1].

II. AGENTSEEKER

AgentSeeker has been developed by following few basic principles: scalability, flexibility, and accurate management of textual documents. In order to consider documents not only as alphanumeric sequences but also as knowledge with a precise meaning, several solutions use meta-tags in order to semantically describe their content. Nevertheless, if we consider as source of information Internet or large local document *corpora*, the tagging of these resources become very complicated due to the impossibility of modifying a file or to the objective difficulty to manually catalogue thousands of documents. For these practical reasons, AgentSeeker can only manage the textual content. Ontologies are used to describe the knowledge of the user, his competences, and his expectations in order to apply them during the document search.

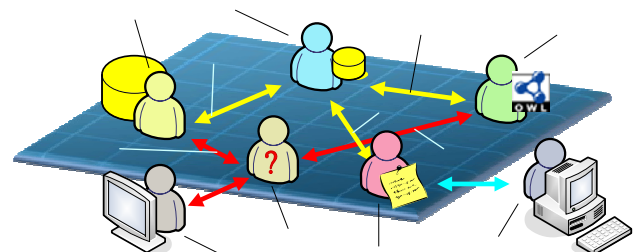


Figure 1: The agent roles presently implemented in AgentSeeker.

AgentSeeker is essentially based on *AgentService* [2], a framework for the development and execution of multi-agent systems, implemented in the C# programming language and by using the Microsoft .NET libraries. An overview of the features and the programming paradigm provided with

AgentService are detailed in [2] and [3].

A. Designing the AgentSeeker overall system

Following the agent oriented paradigm, AgentSeeker is designed as a society of interoperating agents. All the involved components are modeled as agents playing a specific role within the community. Figure 1 shows the different roles and their interactions. In addition to the usual AgentService agents, equipped with *behaviours* and *knowledge objects* (here referenced as *internal agents*), AgentSeeker includes also *external agents*: namely external applications which act like ordinary agents for easily interacting with the rest of the platform. Finally, in order to make agents act in a coordinated way, a set of interaction protocols are defined for modeling their interoperations.

The Indexing Agent

The *Indexing Agent* (IA) is the core of the system. The main goal of the IA is to index documents and inform the *Merger* and the *Manager Agent* about the completion of its work. The documents to be indexed are collected from web sites or from local storages, both distributed on an intranet and stored in a single PC. The IA is able to extract text from common *html* files, simple text documents (*txt*), *pdf* files, and all the formats of the *Microsoft Office Suite*. Parsing hypertexts, the agent extracts also the hyperlinks and distinguishes from *internal links* (namely pages which belong to the same site) and *external links* (pages of other sites). In case of external page, the IA collects the link in a list which will be sent to the manager agent (its features are described below).

IA promotes its services by registering to the yellow pages service included into the system. When IA receives a new job, it first deregisters itself from the yellow pages till the job will be completed, then it renews the registration becoming available again for executing a new job. For each indexing session, IA maintains its own database where stores information extracted from the parsed files (*path*, *content*, *title*, etc.). The local database is based on *.NET Lucene*: the C# porting of a well-known *Apache Foundation* java project named *Apache Lucene* [5]. Essentially developed to store textual contents and to operate queries on them, Lucene is a scalable solution that allows the implementation of large architectures.

The session index is then accomplished in collaboration with the *Merger Agent*, as shown in the next paragraph.

The Merger Agent

Once the IAs have created and stored their partial indexes, there is the need to merge them in order to speed up the search operation by avoiding fragmentations. The *Merger Agent* role (MA) has been defined in order to collect the results of the IAs once they have finished their indexing session. The MA manages a central index (based on Lucene) where the user's queries are materially executed. For this reason the MA has two main tasks: to maintain a central repository of the indexed texts and to respond to the queries coming from the query

agent.

Considering the relation between the dimension of the index and the search speed, *AgentSeeker* provides the possibility to configure the system for involving more than one MA.

The Query Agent

The *Query Agent* (QA) manages the search requests coming from the users through the *Web Interface Agent* and the *Administration Console Agent* or from different external applications. Once a request is received, the QA checks if the ontological support has been requested; in this case, it contacts the *Ontology Agent* in order to have a semantic support for improving the results. Then, it sends a request to all the MAs and collects the consequent results. Finally, it furnishes the results to the web-based interface agent.

Mastering the enterprise's knowledge through the Ontology Agent

The *Ontology Agent* (OA) is the keeper of the knowledge of the system. Its functionalities will be fully described in the section IV but, as an introduction, the OA is essentially able to read ontologies in the OWL language, thanks to the libraries *SemWeb* and *Linq to RDF*. The OA extracts the described concepts and finds the relations among them. On the basis of this information, the OA extends the query sent by the QA, during a user's session.

Another feature of the OA is the classification of the document content. In particular, this service is used by the IA during its indexing sessions, which then receives an estimate of the arguments dealt in the examined text.

The Manager Agent

The *Manager Agent* plays the role of orchestrator, coordinating the activities of the other agents. It is in charge of distributing and balancing the workload among the agents and it acts as supervisor monitoring the index and search processes. In particular the manager has a knowledge object containing the list of web sites (on shared folders) to parse. This list can be increased by adding new sites received from the external agent representing the administration console and by receiving new links discovered by the IAs. In presence of new links to visit, the manager searches for a free IA, consulting the yellow pages. Due to the fact that the yellow pages are distributed across the whole federation, the manager is able to find free agents running also on remote computers. The computational workload is then naturally balanced on every machine and every agent.

The agent-based web interface

From the user's point of view, *AgentSeeker* is a simple web application with a look-and-feel similar to the usual search engines. In the back-end of the web application, an agent is in charge of interacting with a remote *AgentSeeker* installation in order to submit the queries. The interoperation between *AgentSeeker* and the web application is based on a web

service interface exposed by *AgentService*. The life-cycle of this agent is tied with the user's session; every user has his own agent which helps him to interact with the platform. The choice of implementing the web application as an agent simplifies the development of the whole system and integrates the user's interface with the rest of the platform. From the web page, the user can select or import an ontology which represents the argument he has in mind during his query session.

The Administration console

Similar to the previous one, another agent runs behind an administration console which allows administrators to manage AgentSeeker. For example, an Administrator can submit a new web site to index, set the standby time for the platform, or he can directly shut down the platform, stopping safely every agent instance. He can also monitor the status of the platform, namely the agent health, the progression of the indexing tasks, etc.

III. THE FEDERATION MANAGEMENT SUITE

The simplest deployment of AgentSeeker consists in a single platform (in execution on a single computer) with single instances of each agent role. A manager sends jobs to the unique IA, which parses each web site (or folder), classifying every page with the help of the OA. The MA collects the results of the IA, while the QA directly speaks with the external agent behind the web application and with the OA in order to extend the query. A console agent manages the platform.

If the computer has enough resources, the platform administrator could create different instances of the IA in order to process in parallel several jobs. This is particularly useful if the CPU is multi-core, considering also that every IA alternates processing time and downloading of documents.

In case of large amount of textual documents to index, it could be useful to add further computational resources. A new computer is then connected to the first one, a new AgentService platform is installed and new IAs are instanced. The unique manager agent has now at its disposal new IAs which can be contacted through the distributed yellow pages, in a completely transparent way with no complications due to the distributed environment.

Now, with different instances of IAs, only one MA could be not enough. In this case, a new MA can be instanced and the IAs can be instructed in order to refer to a particular MA. With multiple MAs, the QA can submit the query in parallel and then compose the incoming results.

If the catchment area is wide, the federation could be integrated with several instances of query and ontology agents in order to serve different users at the same time.

At this point the scenario can be configured in various ways, with resources totally dedicated to a single type of agent, and mixed platforms with various agent roles. The single computer platform is now spread on a distributed

network, in a totally transparent way from the point of view of the AgentSeeker developer and especially of the system administrator. Furthermore, new computational resources and agent instances can be added or removed dynamically during the AgentSeeker execution.

A. Managing a distributed environment

As shown before, it is plausible a complex federation of platforms with a common goal. In this case, AgentSeeker becomes a distributed society of agents which must act and react in unison. From the point of view of the administrator, it is very important to manage the whole system in a centralized way. Although the pseudo-agent playing the role of *Administration Console* represents a useful tool for tuning some parameters of AgentSeeker and monitoring the activity of the running agents, it is however an entity of the platform with no power on the life-cycle of peers or even of the entire platform.

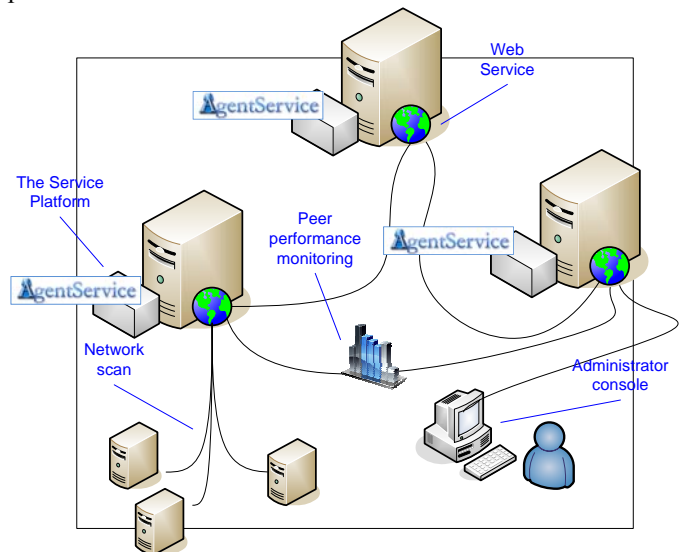


Figure 2: The management infrastructure of a federation.

For this reason we developed an infrastructure which allows a system administrator to easily manage distributed AgentService applications, both at the level of platforms and of agents. AgentSeeker represents a significant test-bed thanks to its intrinsic distribution of resources and scalability.

This infrastructure is a sort of cloud for MASs which enables the administrator to:

- 1) Install a platform on a PC
- 2) Join the platform to a particular federation of platforms
- 3) Discover new platform nodes
- 4) Deploy an application on one or more distributed platforms
- 5) Manage the platform life-cycle
- 6) Create, execute, move, and stop single agent instances.

B. A platform as a service

Fig 2 shows the topology of a federation of AgentService platforms. Each platform is, first of all, a service running on a computer. Installed as a *Windows Service*, each platform is instantiated at the computer wake up. The platform service

exposes a *web service* through which an administrator can set the state of the platform: *idle*, when it is not yet launched; *ready*, when every platform module is loaded; *running*, when the platform is ready to execute agents; *stopping*; and *shuttingdown*.

Every platform-service runs a thread which is in charge of discovering new nodes on the same subnet. Every IP is scanned in order to check if a possible peer platform is listening. In this case, a handshake procedure starts in order to determine if the candidate is available for joining the federation (on the same subnet several federations can coexist and every platform is set up to join one, more, or all the federations). Periodically every node polls the federated platforms to check their existence and to share the list of discovered peers. In this way, the federation is updated, thanks to the interactions of the federated nodes. From the practical point of view, the list of federated platforms is used by the AgentService messaging module to physically route the messages to remote agents.

C. The management of a federation

Once the platforms are physically installed on a network, the administrator can manage the federations connecting its administration GUI to the web service exposed by whichever remote platform. This platform represents the access point of the federation and all the commands coming from the administration GUI transit through it. From the GUI, the administrator is able to see the node list and the status of the related platforms. Further information, constantly updated, regard the CPU occupancy, the available RAM, and, if the platform is running, the number of hosted agents.

The administrator can upload to a single node, or automatically to the whole federation, a set of files like assemblies containing the templates of agents, configuration files, etc. In this manner, a new multi-agent application can be rapidly deployed and executed.

D. Add an agent to the federation

During the usual execution of a distributed multi-agent application as AgentSeeker, it is possible to add new agent instances in order to increase the available resources in the federation. Moreover, it could happen that an agent crashes for some reason and must be stopped and replaced. In these cases, the administrator must be able to create new instances of agents. A first possibility is to select a particular node and then to launch a new instance. Another way is to select the whole federation and create an agent instance suggesting no physical destination. Every node of the federation knows the status of each peer node and is able to find the platform with more free resource. In order to do it, the node classifies the peer performances (CPU time, free RAM, and number of agents) evaluated on a time slot (typically 10 minutes). Once the best platform is selected, the current node checks if the destination has the necessary assemblies containing the agent template and related classes and only if the platform candidate satisfies all the requirements, the agent is materially

instantiated on that node.

E. Moving agents instances

A federation of platforms is an environment where agent activities and interactions evolve in time. An agent could suddenly increase its needs of resources or densely interact with a remote agent. In these cases, the presence of the agent in the current platform could deteriorate the node performances or saturate the network link with a high throughput of messages. As described in [6] the AgentService team developed a facility for moving agents from a platform to another one, saving their state and then resuming them on the destination. The agent state is represented by its collection of knowledge objects which are persisted and sent to the destination along with the needed assemblies.

Considering the GUI we are describing, the administrator is able to select an agent and, evaluating the performances of the other nodes, he can stop it, select the destination, move the agent and resume its activities on the new node, balancing the computational load of the federation.

IV. REPRESENTING KNOWLEDGE IN AGENTSEEKER

Ontological models of the discourse domains which AgentSeeker deals with, allow a sensible growth of the multi-agent application performances.

The aim is to help the user during the submission of a query, taking into account the argument he is considering in order to automatically add more details to the interrogation submitted into the system.

Based on OWL ontologies, AgentSeeker, in its first functioning prototype, provides three policies which exploit the explicit semantic representation of the enterprise's knowledge.

A. A priori classification

This strategy is applied during the indexing sessions and requires a strict interaction between the IA which extracts the text and the OA which classifies it with the help of its knowledge represented by ontologies.

The idea is to estimate the affinity of each processed text with the topics modeled in the ontologies directly supported by AgentSeeker. The MAS maintains a repository where it stores the core ontologies. Since AgentSeeker is able to manage the OWL language, it can accept further ontologies imported by the users. For this reason we can state that AgentSeeker is not tuned on a particular set of ontologies, but it is up to deal with every ontology.

From the practical point of view, every document stored in the Lucene index has a particular field where the URIs of the supported ontologies are related to an estimate of its relevance in respect with the supported semantic models. The measure is expressed in term of percentage of document words which are also described in the ontology. We apply the Porter Stemmer algorithm [7] in order to extract the root of each term suppressing any suffix (plurals, gerundive forms, etc.) and we remove the so-called *stop words* (adverbs, conjunctions, etc.)

from the document, because of their irrelevance.

Once the classification has been completed, the user can order the results on the basis of their relevance according to the arguments supported by AgentSeeker. Usual queries and selection of documents by arguments can be integrated.

B. Conceptual classification

In AgentSeeker we can exploit ontologies in order to classify documents on the basis of a natural hierarchy suggested by the relations of specialization and generalization among concepts. The user suggests the depth of the sub-cluster hierarchy in order to avoid a too detailed classification.

Following this policy, the QA obtains from the OA a hierarchical structure whose nodes represents both the concepts and the single queries to submit to the MA and then to Lucene.

Furthermore, it is possible to restrict the number of processed documents, by clustering only the results of a usual query.

C. Query expansion

The third strategy is aimed to expand the user's query. The OA parses the query in order to add alternatives or more details. According to the argument the user is considering (namely the ontology), the QA analyzes each query term in order to check if it is also an ontological concept. In this case the term can be expanded by following up to three types of policies. The first policy integrates the query, adding, for each word which occurs also in the ontology, all the specialized concepts.

For example, if the user's query is *car retailer* and *car* is an *automobile ontology* concept which is specialized in *station wagon*, *coupe*, and *convertible*, the query is rewritten in this manner: (*station wagon retailer*) OR (*compact retailer*) OR (*coupe retailer*) OR (*convertible retailer*), allowing the user to access also these pages where the term *car* is not explicitly cited. Another type of integration similarly extends the query to those terms which are related to the query keywords through properties (*owl:ObjectProperty*).

Furthermore, each keyword can be integrated by suggesting possible synonyms specified in the given ontology. For this reason we use the owl constructs *owl:sameAs* and *owl:equivalentClass*. Incidentally, this third type allows, potentially, the multi-language support, if the concepts are translated in several languages.

V. CONCLUSIONS AND FUTURE WORKS

At now, AgentSeeker is a fully working prototype, subject to several improvements in term of usability and performances. Moreover, it represents a platform on which we can build specific applications that require large textual repositories to process. Integrating a new application in AgentSeeker is a relatively simple process, because it is necessary only to add a platform (or just agents to the federation) and to interact with the usual AgentSeeker agent roles. The Federation Management Suite ensures a

comfortable tool for managing large deployments of platforms, relieving the administrator from any effort for balancing the computational load. Besides the flexibility ensured by agent-oriented architectures, we can exploit also their intrinsic scalability and adaptability, making AgentSeeker able to tune itself to different contexts: from a little academic laboratory which wants to manage its collection of papers, to the large enterprise which wants to keep the lid on its document corpus.

We use ontologies in order to formally describe the domains where AgentSeeker is called to operate. Presently, the ontology utilization can be considered basic and subject to further improvements. For example we could develop a behaviour for our ontology agent able to reason about the concepts and their relations, in order to find implicit associations and properties. Moreover, explicit properties are now considered as simple links between two concepts; a future improvement will enable the ontology agent to consider, in some way, the meaning of the property.

We plan to introduce also the possibility to explore the web, indexing only those sites which are relevant considering the ontologies included in the AgentSeeker repository. An indexing agent will visit few pages and then ask the ontology agent to determine if the web site is relevant.

In conclusion, we think that AgentSeeker contributes to the improvement of search engine performances, combining a multi-agent system with ontological representations. By using Lucene.NET and homemade spiders, AgentSeeker covers the whole process, from the document parsing to the storage of extracted data. This feature assures full control of every aspect, in respect to other solutions which implement meta-search engines leaning on results of online search engines operations. The solution we propose is then more pragmatic and voted to limit the user's and the administrator's efforts in order to deploy a system which could be used in the everyday work (or life) activity.

REFERENCES

- [1] A. Passadore, A. Grosso, A. Boccalatte, "Indexing enterprise knowledge bases with AgentSeeker", WOA 2009, From Objects to Agents, Parma, Italy, July 2009.
- [2] C. Vecchiola, A. Grosso, A. Passadore, and A. Boccalatte, "AgentService: A Framework for Distributed Multi-agent System Development," to be published in *International Journal of Computers and Applications*, ACTA Press, 2009.
- [3] C. Vecchiola, A. Grosso, and A. Boccalatte, "AgentService: a framework to develop distributed multi-agent systems," *International Journal of Agent-Oriented Software Engineering*, vol. 2, no.3 pp. 290 – 323, 2008.
- [4] Foundation of Intelligent Physical Agents (FIPA), Available: <http://www.fipa.org>.
- [5] E. Hatcher, O. Gospodnetić, and M. McCandlessvan *Lucene in action*, Manning Publications Co, Greenwich, 2009.
- [6] A. Boccalatte, A. Grosso, and C. Vecchiola, "Implementing a Mobile Agent Infrastructure on the .NET Framework," in *Proc. 4th International Conference in Central Europe on .NET Technologies*, Plzen, Czech Republic, May, 2006.
- [7] C. J. Rijsbergen, S. E. Robertson, and M.F. Porter, *New models in probabilistic information retrieval*, British Library, chap. 6, London, 1980.

WADE – An Open Source Platform for Workflows and Agents

G. Caire, E. Quarantotto, G. Sacchi

Telecom Italia
Via Reiss Romoli 274
10148 Torino - Italy

Categories and Subject Descriptors

I.2.11 {**Artificial Intelligence**}: Distributed Artificial Intelligence - *Multiagent systems*; C.2.4 {**Computer Communication Systems**}: Distributed systems; D.2.11 [**Software Engineering**]: Software architecture

General Terms

Management, Performance, Languages.

Keywords

Software Agent, workflow, JADE, Open Source, XPDL, Scalability, Flexibility.

1. INTRODUCTION

WADE is the main evolution of JADE and adds to it the ability to define system logics according to the workflow metaphor.

WADE is not just an add-on, but a complete platform (built on top of JADE) providing advanced administration and fault tolerance mechanisms and enabling a group of agents to cooperatively execute complex tasks defined as workflow.

Nowadays workflows are mostly adopted in BPM (Business Process Management) environments where they are used to represent business processes and orchestrate existing systems typically (but not necessarily) accessible by means of Web Services-based interfaces.

The main challenge in WADE is to bring the workflow approach from the business process level to the level of system internal logics. That is, even if in principle it could be used for that purpose too, WADE does not target high level orchestration of services provided by different systems, but the implementation of the internal behaviour of each single system. Each agent embeds a micro-workflow engine and a complex process can be carried out by a set of cooperating agents each one executing a piece of the process.

The approach followed by WADE is to provide a workflow view on top of a normal Java class. That is a workflow is implemented as a Java class with a well defined structure. A key element in this approach is WOLF (Workflow LiFe cycle management environment), the graphical development environment for WADE based applications.

The presentation will provide an overview of the WADE platform, will highlight its most distinguishing features such as workflow inheritance, support for web services and delegation.

2. WADE OVERVIEW

WADE (Workflow and Agent Development Environment) is a domain independent platform, built on top of JADE 5, an open source middleware for the development of distributed applications based on the agent-oriented paradigm. The distribution of JADE includes a runtime environment, a library of classes that programmers can use to develop their application and some graphical tools for administration and monitoring purposes.

Each running instance of the JADE runtime environment is called Container and a set of containers is called Platform. In a JADE Platform a single special Main Container must always be active and the other containers register with it at startup.

One or more application agents can be started into a Container. The actual job of an Agent is to perform some tasks assigned to it. In JADE, a “Behavior” represents a task to be performed by an Agent and it is implemented as an object of a class that extends the class `Behaviour` of the JADE library.

An Agent to perform its tasks may need to communicate with other Agents in the Platform. JADE provides the agents with the ability to communicate. The communication model adopted is the “Asynchronous Message Passing” and the format of the messages is the ACL (Agent Communication Language) defined by FIPA 5.

WADE adds to JADE the support to the workflow execution and a few mechanisms to manage the complexity of the distribution, in terms of administration and fault tolerance. It should be noticed that a WADE-based application may even not use Workflow Engine agents at all and just exploit the administration and fault tolerance features. In that case we end up with a WADE-based application that does not use workflows. On the other side, it is also possible to use the workflow metaphor inside of a Jade platform.

Wade specific components are:

- **BootDaemon processes:** there is a bootDaemon process for each host in the platform and it is in charge of the Containers activation in its local host.
- **Configuration Agent (CFA):** the configuration agent always runs in the Main Container and is responsible for interacting with the boot daemons and controlling the application life cycle.
- **Controller Agents (CA):** there is a controller agent for each container in the platform and they are responsible for supervising activities in the local container and for all the fault tolerance mechanisms provided by WADE.

- **Workflow Engine Agents (WEA):** Workflow Engine Agents embed an instance of the micro workflow engine and therefore they are able to execute workflows.

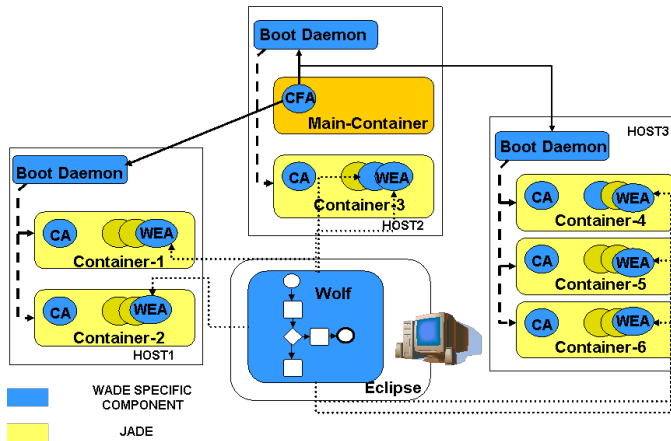


Figure 1 – a WADE platform

Figure 1 shows the topology of a WADE-based application. WADE specific components are highlighted in blue.

As mentioned, workflows are represented in Wade as Java classes, so, in principle, Wade supports “notepad-programming” in the sense that there is not hidden stuff that developers can’t control. However, especially considering that one of the main advantages of the workflow approach is the possibility of representing processes in a friendly graphical form, Wade comes with a development environment called Wolf that facilitates the creation of Wade-based applications. Wolf is an Eclipse plug-in and as a consequence allows Wade developers to exploit the full power of the Eclipse IDE plus additional Wade specific features.

3. Workflow approach

A workflow is a formal definition of a process in terms of activities to be executed, relations between them, criteria that specify the activation and termination and additional information such as the participants, the software tools to be invoked, required inputs and expected outputs and internal data manipulated during the execution.

The key aspect of the workflow metaphor is the fact that the execution steps as well as their sequencing are made explicit. This makes it possible to give a graphical representation of a process defined as a workflow. Such representation is clearly extremely more intuitive with respect to a piece of software code and in general is understandable by domain experts as well as by programmers.

Domain experts can therefore validate system logics directly and not only on documents that most of the time are not perfectly up to date. In some cases they could even contribute to the actual development of the system without the need for any programming skill.

Another important characteristic is that, being the execution steps explicitly identified, the workflow engine (i.e. a system able to automatically execute a process defined as a workflow) can trace them. This makes it possible to create automatic mechanisms to facilitate system monitoring and problem investigation.

Additionally, when processes have to be executed within the scope of a transaction, semi-automatic rollback procedures can be activated in case of unexpected fault. Finally, since workflows are fully self-documented, workflow-based development releases the development team of the burden of keeping documentation aligned each time design choices must be revisited to face implementation details or evolving requirements.

Nowadays the workflow metaphor is mostly used in BPM environments where a workflow represents a business process and orchestrates a number of existing systems typically (but not necessarily) accessible by means of Web Services based interfaces.

The main challenge in WADE is to bring the workflow approach from the business process level to the level of system internal logics. That is, even if it could be used for that purpose too, WADE does not target high level orchestration of services provided by different systems, but the implementation of the internal behaviour of each single system.

First of all it should be noticed that WADE does not include a single powerful workflow engine as the majority of BPM oriented tools. On the contrary WADE provides an extension of the basic Agent class of the JADE library called `WorkflowEngineAgent` that embeds a small and lightweight workflow engine (we talk about “micro-workflow engine”). As a consequence, besides normal JADE behaviours, all Workflow-Engine agents active in a WADE-based multi-agent applications are able to execute workflows represented according to a WADE specific formalism.

The second important point to highlight is that, in order to allow developers to exploit the workflow metaphor to define system internal logics and, at the same time, to give them the same power of a software programming language and a comparable execution efficiency, the WADE workflow representation formalism is based on the Java language. That is, a workflow that can be executed by WADE Workflow-Engine agents is expressed as a Java class with a well defined structure (as it will be explained in the following). As such WADE workflows can be edited, refactored, debugged and in general managed as all Java classes and can include all pieces of code (methods, fields of whatever types, inner classes, references to external classes and so on) needed to implement the process details. In addition, of course, the execution flow they specify can be presented and modified in a friendly, graphical way. More in details WOLF (the development environment for WADE based applications) is an Eclipse plugin and allows developers to work with a graphical view (suitable to manage the process flow) and a code view (the usual Eclipse Java editor suitable to define execution details) that are kept in synch.

Finally it must be noticed that WADE does not impose that all system logics are defined as workflows. Developers are free to exploit the workflow metaphor to describe those tasks for which they think it is appropriate and use normal JADE behaviours (or other purely Java patterns) elsewhere.

As mentioned the approach followed by WADE is to provide a workflow view on top of a normal Java class.

As a consequence, a workflow is implemented as a Java class and no standard workflow definition language is used. However, Wolf adopts the workflow meta-model defined in the XPDL [2],

standard specified by the Workflow Management Consortium. The XPD L meta-model has been chosen, because the XPD L language has been conceived as interchange formalism between different systems. WADE supports the import of XPD L files and the adoption of this meta-model facilitates these operations. Moreover, the XPD L meta-model is based on a Finite State Machine computational model that is the same model supported by the WADE agents.

In the XPD L meta-model a process is represented as a workflow, consisting of one or more **activities** that can be thought as tasks to be executed.

In a workflow, the execution entry point is defined, specifying the first activity to be performed; this activity is called **Start Activity**. On the other hand, a workflow must have one or more termination points, named **Final Activities**.

The execution flow is defined by means of **transitions**. A transition is an oriented connection between two activities and may have a condition associated. Regular or exception transitions can be defined. Exception Transitions allow specifying branches that are taken only when an Exception is raised in the source activity.

Excluding the final ones, each activity may have one or more outgoing transitions. When the execution of an activity is terminated, the conditions associated to its outgoing transitions are evaluated. As soon as a condition is verified the corresponding transition is activated and the execution flow proceeds towards the destination activity.

Normally a process execution uses some internal data, for instance, to pass intermediate results between activities and/or for evaluation of conditional expressions. In the XPD L meta-model internal data are modeled by **Data Fields**.

A process can have one or more inputs to be provided and one or more outputs expected at the end of its execution. Inputs and outputs of a process can be formalized in the XPD L meta-model by means of the workflow **Formal Parameters**.

The XPD L meta-model defines some predefined types of activity. The most important ones are:

- **Tool Activity**: a tool activity is an activity that is implemented by means of the invocation of one or more software tools, named **Applications**.
- **Subflow Activity**: a subflow activity is an activity that requires the execution of another workflow. When a workflow is called by a subflow activity, the workflow formal parameters permit the exchange of necessary data between calling and called process. A distinguishing characteristic of the WADE workflow engine is the **Delegation mechanism** that allows a set of agents to cooperatively execute a complex process. More in details the agent executing the calling workflow can decide to delegate the subflow to another agent on the basis of conditions evaluated at runtime. Such conditions may be related for instance to the current load (thus supporting the implementation of a GRID-like system) or to specific abilities required to carry out a portion of the whole process. The delegation mechanism is implemented by means of an extension of the fipa-contract-net protocol (5) where the agent executing the calling workflow acts as initiator while

the agent executing the subflow acts as responder. This protocol allows managing, when required, a process carried out by a set of cooperating agents as a single transaction.

- **Route Activity**: a route activity is an activity which performs no work processing, but simply supports routing decisions among its incoming and out coming transitions.

Finally, WADE introduces a few types of Activity, among them the **CodeActivity** and the **Web Services Activity**.

In a **Code Activity** the operations are specified directly by a piece of Java code embedded in the workflow process definition.

A **Web Service Activity**, instead, provides an easy way to invoke Web Services from a workflow. Two kinds of Web Services invocations are allowed: static and dynamic.

In the first case, it is necessary to import the WSDL describing the Web Service using WOLF. This operation generates a set of classes that will be used at workflow execution time to actually invoke the web service described by the imported WSDL. These classes can be extended by users with development skills, in order to satisfy some specific requirements (e.g.: logging some information before and after the invocation of the Web Service).

In the second case, the dynamic invocation is performed without the need of generating new classes. This approach is useful if the user doesn't need to customise these classes.

As mentioned a workflow is implemented by a Java class and it extends directly or indirectly the WADE class `WorkflowBehaviour`.

The `WorkflowBehaviour` class provides a set of APIs, consistent with the XPD L meta-model described in the previous section, which can be used by developers to implement their own workflows.

The mapping between the meta-model objects and the workflow implementation is shown in the example depicted in Figure 2.

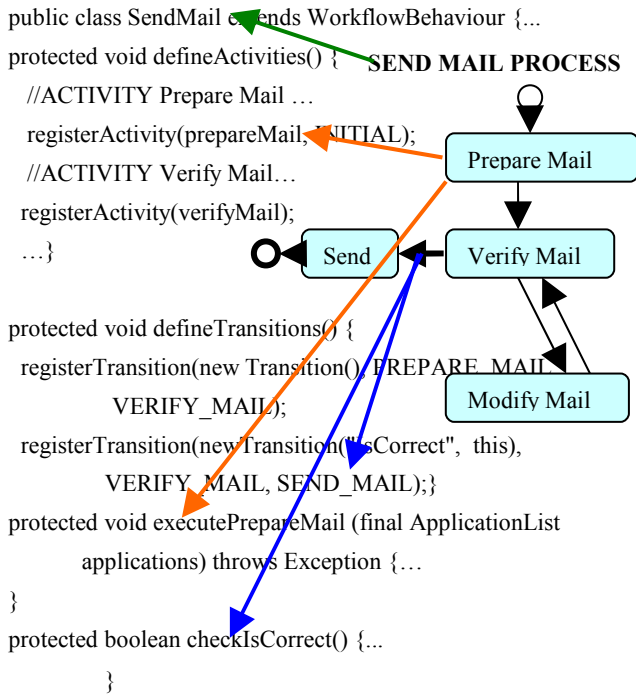


Figure 2 – Mapping between meta – model objects and workflow implementation

The methods `registerActivity()` and `registerTransition()` must be used respectively to add an activity and a transition.

When the `registerActivity()` method is called, its first parameter is an instance of the activity to be added.

The actual tasks to be performed by an activity (no matter of its type) are specified in a `void` method of the workflow class; this method must have the same name of the activity, preceded by the prefix “execute” (`execute<ActivityName>`). The workflow engine is in charge of invoking that method when the activity is visited.

In the same way, the `registerTransition()` method takes a transition instance as parameter. For transitions with conditions, the `boolean` expression to be evaluated by the workflow engine is specified in a `boolean` method that has same name of the condition, preceded by the prefix “check” (`check<ConditionName>`).

An important feature of the object-oriented programming languages is the possibility to reuse the code by means of **inheritance mechanisms**. In order to bring the programming languages power also to the workflow representation, it has been chosen to provide the workflow with the inheritance mechanism too. Therefore it is possible to define a new workflow extending an old one, and then adding/removing activities and transitions. The overriding of methods associated to the activities and to the conditions of transitions is also permitted.

4. CONCLUSIONS

In this paper we described WADE, an open source framework to develop distributed applications based on the agent programming paradigm. WADE is based on JADE and in particular adds to it

the possibility of modeling the agents’ behaviours following the workflow metaphor. Moreover some administration and fault tolerance features are provided too.

As mentioned, in WADE a workflow is represented as a java class and WOLF provides a graphical view of it, making available to the developers both the expressiveness of a visual representation and the power of usual programming languages.

As well known, the workflow metaphor is traditionally used in the BPM context for web service orchestration. Because WADE provides support for Web Service invocation, it can be used even in this context, but its actual challenge is to bring the workflow approach from the business process level to the level of system internal logics. A direct consequence of the approach described is that the full power of WADE can be exploited for applications that imply the execution of possibly long and fairly complex tasks.

Furthermore, unlike the majority of existing workflow systems that provide a powerful centralized engine, in WADE each agent can embed a “micro workflow engine” and therefore a complex process can be carried out by a set of cooperating agents through the delegation mechanism.

From an industrial point of view WADE can be particularly useful to develop applications with strong requirements of both performance and scalability and high flexibility in defining the systems’ logics.

5. REFERENCES

- [1] JADE - Java Agent Development framework. <http://jade.tilab.com>
- [2] FIPA – Foundation for Intelligent Physical Agents <http://www.fipa.org>
- [3] FIPA – The FIPA Contract Net interaction protocol
- [4] XPDL XML Process Definition Language, <http://www.wfmc.org/standards/xpdl.htm>
- [5] Shapiro, R. 2002. A comparison of XPDL, BPML and BPEL4WS (Rough Draft), Cape Vision
- [6] BPMN Business Process Modeling Notation <http://www.bpmn.org/>
- [7] G. Caire “WADE: An Open Source Platform for Workflows and Agents” <http://jade.tilab.com/wade/doc/tutorial-aamas2008.zip>
- [8] Caire G., Gotta D., Banzi “WADE: A software platform to develop mission critical applications exploiting agents and workflows”, M., Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008) – Industry and Applications Track, Berger, Burg, Nishiyama (eds.), May, 12-16., 2008, Estoril, Portugal, pp. 29-36.
- [9] “WADE User Guide” <http://jade.tilab.com/wade/doc/WADE-User-Guide.pdf>
- [10] G. Caire, M. Porta, E. Quarantotto, G. Sacchi “Wolf – An Eclipse Plug-in for WADE”

MALLOW Workshop on Coordination, Organization, Institutions and Norms in Agent Systems & On-line Communities (COIN@MALLOW2009)

Julian Padget* and Axel Polleres†

*Department of Computer Science, University of Bath, Bath BA2 7AY, United Kingdom
Email: jap@cs.bath.ac.uk

† Digital Enterprise Research Institute, National University of Ireland, Galway
Email: axel.polleres@deri.org

Abstract

The COIN (Coordination, Organizations, Institutions and Norms) workshop series began in 2005 with a single satellite workshop at AAMAS in Utrecht [1]. Since then two meetings have been organized each year, with one typically attached to AAMAS and the other to another major conference in a different continent from AAMAS [2], [3], [4]. This year, exceptionally, a third event is being held in support of the MALLOW federated workshops. Each meeting—particularly those not associated with AAMAS—aims to build links to an associated research theme to explore and encourage opportunities for cross-fertilization.

I. INTRODUCTION

For this edition of COIN, the focus is on how COIN topics influence and are realized in on-line communities, where it is necessary to take into account social, legal, economic and technological dimensions of agent-agent, agent-human, human-human interactions in order to ensure social order within these environments. Such communities are dynamic and offer significant challenges because of their potential dissimilarity to conventional human social structures, with the associated strengths, weaknesses, opportunities and threats posed by alternative interaction modalities that derive from the different physics of such virtual environments. This leads to the exploration and establishment of new normative frameworks that do not necessarily have parallels in the physical world but are well reflected in “made natures” such as the Web. Such frameworks are not only interesting for the agent community but also have attracted interest of the Semantic Web Community in events such as the recent W3C Workshop on the Future of Social Networking.

The accepted presentations reflect a mix of both classical COIN topics and theme-related papers. Highly interactive sessions are a feature of the COIN meetings that we try to enable through a range of alternative approaches: in this case one session will comprise regular papers where each of these papers is introduced by one of the other presenters; a second session features an invited talk from Alexandre Passant who is closely involved with the W3C Social Web Incubation Group, while the third session will function as a round table, driven by several short presentations of work in progress, highlighting open issues and again related to the on-line communities theme.

There are five regular papers, each of which received at least three reviews in order to supply the authors with comprehensive feedback, and four work-in-progress/project status papers. Selected and extended papers will appear in the annual COIN volume in the Springer LNCS series. We would like to thank all authors for their contributions and the members of the Programme Committee for the excellent work during the reviewing phase.

Julian Padget, Axel Polleres
August 11, 2009

II. WORKSHOP COMMITTEES

Workshop Organizers

Julian Padget	University of Bath, UK
Axel Polleres	National University of Ireland, Galway

Programme Committee

Alexander Artikis	Demokritos Research Centre, Greece
Sören Auer	University of Leipzig, Germany
Guido Boella	Università degli Studi di Torino, Italy
Frances Brazier	Vrije Universiteit Amsterdam, Netherlands
Dan Brickley	FOAF Project
John Breslin	National University of Ireland, Galway
Antonio Carlos Costa	Universidade Catolica de Pelotas, Brazil
Stephen Cranefield	University of Otago, New Zealand
Harry Halpin	University of Edinburgh, UK
Jomi Fred Hübner	ENS Mines Saint-Etienne, France
Lloyd Kamara	Imperial College London, UK
Eric Matson	Purdue University, USA
Pablo Noriega	IIIA-CSIC, Spain
Eamonn O'Neill	University of Bath, UK
Alexandre Passant	National University of Ireland, Galway
Jeremy Pitt	Imperial College London, UK
Juan Antonio Rodriguez Aguilar	IIIA-CSIC, Spain
Sascha Ossowski	Universidad Rey Juan Carlos, Spain
Sebastian Schaffert	Salzburg Research, Austria
Jaime Sichman	University of São Paulo, Brazil
Maarten Sierhuis	RIACS/NASA Ames Research Center, USA
Kostas Stathis	Imperial College London, UK
Harko Verhagen	Stockholm University/KTH, Sweden
Niek Wijngaards	D-CIS Lab (Thales), The Netherlands

COIN Steering Committee

Guido Boella	University of Torino, Italy
Olivier Boissier	ENS Mines Saint-Etienne, France
Nicoletta Fornara	University of Lugano, Italy
Christian Lemaître	Universidad Autónoma Metropolitana, Mexico
Eric Matson	Purdue University, USA
Pablo Noriega	IIIA-CSIC, Spain
Sascha Ossowski	Universidad Rey Juan Carlos, Spain
Julian Padget	University of Bath, UK
Jeremy Pitt	Imperial College London, UK
Jaime Sichman	University of São Paulo, Brazil
Wamberto Vasconcelos	University of Aberdeen, UK
Javier Vázquez Salceda	Universitat Politècnica de Catalunya, Spain
George Vouros	University of the Aegean, Greece

III. LIST OF PAPERS

Full Papers:

- Normative Programming for Organisation Management Infrastructures
Jomi Fred Hübner, Olivier Boissier and Rafael H. Bordini
- Towards an architecture for self-regulating agents: a case study in international trade
Brigitte Burgemeestre, Joris Hulstijn and Yao-Hua Tan
- Four Ways to Change Coalitions: Agents, Dependencies, Norms and Internal Dynamics
Guido Boella, Leon van der Torre and Serena Villata
- Dealing with incomplete normative states
Juan Manuel Serrano and Sergio Saugar
- Directing Status Messages to their Audience in Online Communities
Milan Stankovic, Alexandre Passant and Philippe Laublet

Project Descriptions:

- Managing water demand as a regulated open MAS
Vicent Botti, Antonio Garrido, Adriana Giret and Pablo Noriega
- Open Interaction System Specification and Monitoring Using Semantic Web Technology
Nicoletta Fornara
- SIOC Project: Semantically Interlinked Online Communities
Uldis Bojars and Alexandre Passant
- Norms, Organisations and Semantic Web Services: The ALIVE approach
Sergio Alvarez-Napagao, Owen Cliffe, Julian Padget and Javier Vázquez

IV. SPONSORING INSTITUTIONS

Axel Polleres' work is supported by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2)

REFERENCES

- [1] O. Boissier, J. Padget, V. Dignum, G. Lindemann, E. Matson, S. Ossowski, J. Simão Sichman, and J. Vázquez-Salceda, Eds., *Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems: AAMAS 2005 International Workshops on Agents, Norms and Institutions for Regulated Multi-Agent Systems, ANIREM 2005, and Organizations in Multi-Agent Systems, OOP 2005, Revised selected papers.*, ser. LNCS, vol. 3913. Springer Verlag, June 2006, ISBN: 3-540-35173-6. Available via <http://dx.doi.org/10.1007/11775331>.
- [2] J. Vazquez-Salceda and P. Noriega, Eds., *Coordination, Organizations, Institutions and Norms, COIN 2006, Hakodate, Japan, May 8–9, 2006 and Riva del Garda, Italy, August 21–22, 2006, Revised Selected and Invited Papers*, ser. Lecture Notes in Computer Science, vol. 4386. Springer, 2007.
- [3] J. S. Sichman, J. A. Padget, S. Ossowski, and P. Noriega, Eds., *Coordination, Organizations, Institutions, and Norms in Agent Systems III, COIN 2007, International Workshops COIN@AAMAS 2007, Honolulu, HI, USA, May 14, 2007, COIN@MALLOW 2007, Durham, UK, September 3-4, 2007, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 4870. Springer, 2008.
- [4] J. F. Hübner, E. Matson, O. Boissier, and V. Dignum, Eds., *Coordination, Organizations, Institutions and Norms in Agent Systems IV COIN 2008 International Workshops, COIN@AAMAS 2008, Estoril, Portugal, May 12, 2008. COIN@AAAI 2008, Chicago, USA, July 14, 2008. Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 5428. Springer, 2009, ISBN 978-3-642-00442-1. Available via <http://dx.doi.org/10.1007/978-3-642-00443-8>.

Normative Programming for Organisation Management Infrastructures

Jomi F. Hübner^{†*}

^{*}Dept Automation and Systems Eng.
Federal University of Santa Catarina
Florianópolis, Brazil
Email: jomi@das.ufsc.br

Olivier Boissier[†]

[†]Ecole Nationale Supérieure des Mines
Saint Etienne, France
Email: {hubner, boissier}@emse.fr

Rafael H. Bordini[‡]

[‡]Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, Brazil
Email: R.Bordini@inf.ufrgs.br

Abstract—Recent work shows a tendency to use programming languages specific to the social aspects of multi-agent systems, for example in programming norms that agents ought to follow. In this paper, we introduce a simple and elegant normative programming language called NPL and show its operational semantics. We then define a particular class of NPL programs that are suitable for programming Organisation Management Infrastructures (OMI) for *MOISE*, defining a Normative Organisation Programming Language (NOPL). We show how *MOISE*'s Organisation Modelling Language can be translated into NOPL, and briefly describe how this all has been implemented on top of an artifact-based OMI for *MOISE*.

I. INTRODUCTION

The use of organisational and normative concepts is widely accepted as a suitable approach for the design and implementation of Multi-Agent Systems (MAS) [1]–[4]. Although these concepts are useful for MAS methodologies and therefore used at design time, in this paper we focus on their use at run-time. We conceive of a multi-agent system as a set of agents participating to an organisation by playing roles in it. An important component of MAS is thus the Organisation Management Infrastructure (OMI), which exists in a system to help and supervise agents in the achievement of the purpose of the organisation.

A recent trend in the development of OMIs is to provide languages that the MAS designer (human or artificial in the case of self-organisation) uses to write a program that will define the *organisational* functioning of the system, complementing agent programming languages that defines the *individual* functioning of the system. The former type of languages can focus on different aspects of the overall system, for example: structural aspects (roles and groups) [5], dialogical aspects [2], coordination aspects [6], and normative aspects [7], [8]. The OMI is then responsible for interpreting such a language and providing corresponding services to the agents. For instance, in the case of *MOISE*⁺ [4], the designer can program a norm such as “an agent playing the role ‘seller’ is *obliged* to deliver some goods after being payed by the agent playing role ‘buyer’”. The OMI is responsible for identifying the activation of that obligation and to enforce the compliance to that norm by the agents playing the corresponding roles.

We are particularly interested in a flexible and adaptable implementation of OMIs. Such implementation is normally

coded using an object-oriented programming language (e.g. Java). However, the exploratory stage of current OMI languages often requires changes in the implementation so that one can experiment with new features. The refactoring of the OMI for such experiments is usually an expensive task that we would like to simplify. Our work therefore addresses one of the main missing ingredients for the *practical* development of sophisticated multi-agent systems where the macro-level requires complex organisational and normative structures in the context of so many different views and approaches still being actively researched by the MAS research community.

This problem is particularly complex for organisation models that consider elements with different natures like groups, roles, common goals, and norms. These elements have their own life cycle, are bound together, and are *constrained* by a set of properties (e.g. role compatibility and cardinality). Our proposal is thus a uniformed approach where all kinds of constraints are expressed by norms. These norms then can be explicitly and flexibly enforced by different mechanisms. The OMI is then mainly concerned with providing such mechanism instead of considering all kinds of constraints. However, we do not want to force the MAS designer to program the organisation using only norms. The designer should program their organisation using more suitable constructors. For example, using a role cardinality constructor to state “a classroom has one professor” instead of a norm like “it is prohibited that two agents play the role professor in the same classroom”).

The solution presented in this paper is to translate a more abstract language into another simpler language. The problem of implementing the OMI is thus reduced to a translation problem, which is usually much simpler and less error prone. We start from an organisational modelling language which is then automatically translated into a normative programming language. The language available to the MAS designer has thus more abstract concepts (such as groups, roles, and global plans) than normative languages. More precisely, our starting language is the *MOISE* Organisation Modelling Language (OML — see Sec. III) and our target language is the Normative Organisation Programming Language (NOPL — Sec. IV). NOPL is a particular class of programs of a normative programming language presented and formalised in this paper (Sec. II). All of this has been implemented on top of our

previous work on OMI where an artifact-based approach, called ORA4MAS, is used (Sec. V).

The main contributions of this work are: (i) a normative programming language and its formalisation using operational semantics; (ii) the translation from an organisational language into the normative language; and (iii) an implemented artifact-based OMI that interprets the target normative language. These contributions are better discussed and placed in the context of the relevant literature in Sec. VI.

II. NORMATIVE PROGRAMMING LANGUAGE

Although several languages for norms are available, (e.g. [7]–[9]), for this project we need a language that handles *obligations* and *regimentation*. While agents can violate obligations (and sanctions might take place later), regimentation is a preventive strategy of enforcement: agents are not capable to violate a regimented norm [10]. Regimentation is important for an OMI to allow situations where the designer wants to define norms that must be followed because its violation represent a serious risk for the organisation.¹ The current languages either consider obligation or regimentation as enforcement strategies, and do not allow the designers (nor the agents) to dynamically choose the best strategy for their application.

Our language can be relatively simple because we do not need prohibitions nor permission as primitives. By default, everything is permitted and thus the designer does not need to code permissions. Prohibitions can be represented either by regimentation or as an obligation for someone else to decide how to handle the situation (this approach is inspired by the approach by Grossi et al. [10]). For example, consider the norm “it is prohibited to submit a paper with more than 6 pages”. In case of regimentation of this norm, tentatives to submit a paper with more than 6 pages will fail. In case this norm is not regimented, the designer has to define a norm such as “when a paper with more than 6 pages is submitted, the chair has to decide whether to accept the submission or not”. Another assumption that allowed us to devise a simple language is that we do not consider inconsistent norms. Either the programmer or the program generator are supposed to handle this issue.

A. Syntax

Given the above requirements and simplifications, we introduce below a new Normative Programming Language (NPL) (Fig. 1 contains the definition of its syntax).² A normative program *np* is composed of: (i) a set of facts and inference rules (as in Prolog); and (ii) a set of norms. A NPL norm has the general form `norm id : φ -> ψ` , where *id* is a unique *identifier* of the norm; φ is a formula that determines the *activation condition* for the norm; and ψ is the *consequence* of

¹The importance of regimentation is corroborated by relevant implementations of OMI, such as AGR, *S-MOISE*⁺, and ISLANDER, which consider regimentation as an important enforcement mechanism.

²The non-terminals not included in the specification, *atom*, *id*, *var*, and *number*, correspond, respectively, to predicates, identifiers, variables, and numbers as used in Prolog.

```

np      ::= “np” atom “{” ( rule | norm )* “}”
rule    ::= atom [ “:-” formula ] “.”
norm    ::= “norm” id “:” formula “->” ( fail | obl ) “.”

fail    ::= “fail(” atom “)”
obl     ::= “obligation(”
           (var | id) “,” atom “,” formula “,” time “)”

formula ::= atom | “not” formula |
           atom ( “&” | “|” ) formula
time    ::= “” ( “now” |
           number ( “second” | “minute” | ...))
           “” [ ( “+” | “-” ) time ]
    
```

Fig. 1. EBNF of the NPL

the activation of the norm. Two types of norm consequences ψ are considered:

- *fail* – `fail(r)`: represents the case where the norm is regimented. Argument *r* represents the reason for the failure;
- *obl* – `obligation(a, r, g, d)`: represents the case where a new obligation has to be created for some agent *a* as the consequence of the norm activation. Argument *r* is the reason for the obligation (which has to include the norm’s *id*); *g* is the formula that represents the obligation (a state of the world that the agent must try to bring about, i.e. a goal it has to achieve); and *d* is the deadline to fulfil the obligation.

A simple example to illustrate the language is given below; we used source code comments to explain the program.

```

np example {
  a(1). a(2). // facts
  ok(X) :- a(A) & b(B) & A>B & X = A*B. // rule
  // note that b/1 is not defined in the program;
  // it is a dynamic fact provided at run-time

  // alice has 4 hours to achieve a value of X < 5
  norm n1: ok(X) & X > 5
  -> obligation(alice,n1,ok(X) & X<5, 'now'+4 hours').

  // bob is obliged to sanction alice in case X > 10
  norm n2: ok(X) & X > 10
  -> obligation(bob,n2,sanction(alice), 'now'+1 day').

  // example of regimented norm; X cannot be > 15
  norm n3: ok(X) & X > 15 -> fail(n3(X)).
}
    
```

As in other approaches (e.g. [11], [12]), we have a static/declarative aspect of the norm (where norms are expressed in NPL resulting in a normative program) and a dynamic/operational aspect (where obligations are created for existing agents). We call the first aspect simply norm and the second obligation. An obligation has thus a run-time life-cycle. It is created when the activation condition φ of some norm *n* holds. The activation condition formula is used to instantiate the values of variables *a*, *r*, *g*, and *d* of the obligation to be created. Once created, the initial state of an obligation is *active* (Fig. 2). The state changes to *fulfilled* when agent *a* fulfils the norm’s obligation *g* before the deadline *d*. The obligation state changes to *unfulfilled* when agent *a* does not fulfil the norm’s

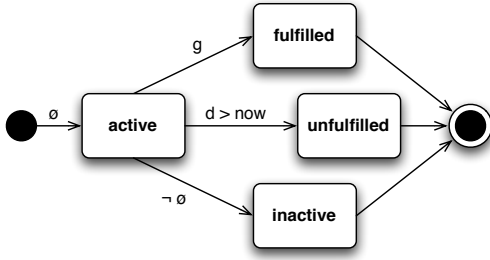


Fig. 2. State Transitions for Obligations

obligation g before the deadline d . As soon as the activation condition of the norm that creates the obligation (φ) ceases to hold, the state changes to *inactive*. Note that a reference to the norm that led to the creation of the obligation is kept as part of the obligation itself (the r argument), and the activation condition of this norm must remain true for the obligation to stay active; only an active obligation will become either fulfilled or unfulfilled, eventually. Fig. 2 shows the obligation life-cycle.

B. Semantics

We now give semantics to NPL using the well known structural operational semantics approach [13].

A program in NPL is essentially a set of norms where each norm is given according to the grammar in Fig. 1; it can also contain a set of initial facts and inference rules specific to the program's domain (all according to the grammar of the NPL language). The normative system operates in conjunction with an agent execution system; the former is constantly fed by the latter with “facts” which, possibly together with the domain rules, express the current state of the execution system. Any change in such facts leads to a potential change in the state of the normative system, and the execution system checks that the normative system is still in a sound state before committing towards particular execution steps; similarly, it can have access to current obligations generated by the normative system. The overall system's clock also causes potential changes in the state of the transition system, by changing the time component of its configuration.

As we use operational semantics to give semantics to the normative programming language (i.e. the language used to program the normative system specifically), we first need to define a configuration of the transition system that will be defined through the semantic rules presented later. A configuration of our normative system, giving semantics to NPL, is a tuple $\langle F, N, \top, OS, t \rangle$ where:

- F is a set of facts received from the execution system and possibly rules expressing domain knowledge. The former works as a form of input from the agent execution system to the normative system. Each formula $f \in F$ is, as explained earlier, an atomic first order formula or a Horn clause.

- N is a set of norms, where each norm $n \in N$ is a norm in the syntax defined for *norm* in the grammar in Fig. 1.
- The state of the normative system is either a sound state denoted by \top or a failure state denoted by \perp ; the latter is caused by *regimentation* through the $\text{fail}(_)$ language construct within norms. This is accessible to the agent execution system which prevents the execution of the action which would lead to the facts causing the failure state, and rolls back the facts about the state of the execution system.
- OS is a set of obligations, each accompanied by its current state; each element $os \in \text{osets}$ is of the form $\langle o, \text{ost} \rangle$ where o is an obligation, again according to the syntax for obligations given in Fig. 1, and $\text{ost} \in \{\text{active}, \text{fulfilled}, \text{unfulfilled}, \text{inactive}\}$ (the possible states of each individual obligation). This is also of interest to the agent execution system and thus accessible to it.
- t is the current time which is automatically changed by the underlying execution system, using, of course, a discrete, linear notion of time. For the purpose of the operational semantics, it is assumed that all rules that apply at a given time are actually applied before the system changes the state to the next time.

Given a normative program P — which is, remember, a set of facts and rules (P_F) and a set of norms (P_N) written in NPL — the initial configuration of the normative system (before the system execution starts) is $\langle P_F, P_N, \top, \emptyset, 0 \rangle$.

In the semantic rules, we use the notation T_c to denote the component c of tuple T . The semantic rules are as follows.

1) *Norms*: The rule below formalises *regimentation*: when any norm n becomes active — i.e. its *condition* component holds in the current state — and its *consequence* is $\text{fail}(_)$, we move to a configuration where the normative state is no longer sound but a failure state (\perp). Note that we use n_φ to refer to the condition part of norm n (the formula between “:” and “ \rightarrow ” in NPL's syntax) and n_ψ to refer to the consequence part of n (the formula after “ \rightarrow ”).

$$\frac{n \in N \quad F \models n_\varphi \quad n_\psi = \text{fail}(_)}{\langle F, N, \top, OS, t \rangle \longrightarrow \langle F, N, \perp, OS, t \rangle} \quad (\text{Regim})$$

The underlying execution system, after realising a failure state caused by Rule **Regim** above, needs to ensure the facts are rolled back to the previously consistent state, which will make the following rule apply.

$$\frac{\forall n \in N. (F \models n_\varphi \Rightarrow n_\psi \neq \text{fail}(_))}{\langle F, N, \perp, OS, t \rangle \longrightarrow \langle F, N, \top, OS, t \rangle} \quad (\text{Consist})$$

The next rule is similar to Rule **Regim** but instead of failure, the consequence is the creation of an obligation. In the rule, ‘m.g.u.’ means “most general unifier” as in Prolog-like unification; the notation $t\theta$ means the application of the variable substitution function θ to formula t . Note that we required that the deadlines of newly created obligations are not

yet past. The notation $\stackrel{\text{obl}}{=}$ is used for equality of obligations, which ignores the deadline in the comparison. That is, we define that an obligation $\text{obligation}(a, r, g, d)$ equals to an obligation $\text{obligation}(a', r', g', d')$ if and only if $a = a'$, $r = r'$, and $g = g'$. Because of this, Rule **Oblig** does not allow the creation of the same obligation with two different deadlines. Note also that if there already exists an equal obligation but it has become inactive, this does not prevent the creation of the obligation.

$$\frac{n \in N \quad F \models n_\varphi \quad n_\psi = o \quad o\theta_d > t \quad \neg \exists \langle o', ost \rangle \in OS . (o' \stackrel{\text{obl}}{=} o\theta \wedge ost \neq \text{inactive})}{\langle F, N, \top, OS, t \rangle \longrightarrow \langle F, N, \top, OS \cup \langle o\theta, \text{active} \rangle, t \rangle} \quad (\text{Oblig})$$

where θ is the m.g.u. such that $F \models o\theta$

2) *Obligations*: Recall that a NPL obligation has the general form $\text{obligation}(a, r, g, d)$. With a slight abuse of notation, we shall use o_a to refer to the agent that has the obligation o ; o_r to refer to the reason for obligation o ; o_g to refer to the state of the world that agent o_a is obliged to achieve (the *goal* the agent should adopt); and o_d to refer to the deadline for the agent to do so. An important aspect of obligation syntax is that the NPL parser always ensures that the programmer used the norm's *id* as predicate symbol in o_r and so in the semantics, when we say o_r , we are actually referring to the activation condition n_φ of the norm used to create the obligation.

Rule **Fulfil** says that the state of an active obligation o should be changed to **fulfilled** if the state of the world o_g that the agent agent was obliged to achieve has already been achieved (i.e. the domain rules and facts from the underlying system imply g). Note however that such state must have been achieved *within the deadline*.

$$\frac{os \in OS \quad os = \langle o, \text{active} \rangle \quad F \models o_g \quad o_d \geq t}{\langle F, N, \top, OS, t \rangle \longrightarrow \langle F, N, \top, (OS \setminus \{os\}) \cup \{\langle o, \text{fulfilled} \rangle\}, t \rangle} \quad (\text{Fulfil})$$

Rule **Unfulfil** says that the state of an *active obligation* o should be changed to **unfulfilled** if the deadline is already past; note that the rule above would have changed the status to **fulfilled** so the obligation would no longer be active if it had been achieved in time.

$$\frac{os \in OS \quad os = \langle o, \text{active} \rangle \quad o_d < t}{\langle F, N, \top, OS, t \rangle \longrightarrow \langle F, N, \top, (OS \setminus \{os\}) \cup \{\langle o, \text{unfulfilled} \rangle\}, t \rangle} \quad (\text{Unfulfil})$$

Rule **Inactive** says that the state of an active obligation o should be changed to **inactive** if the reason (i.e. motivation) for the obligation no longer holds in the current system state reflected in F .

$$\frac{os \in OS \quad os = \langle o, \text{active} \rangle \quad F \not\models o_r}{\langle F, N, \top, OS, t \rangle \longrightarrow \langle F, N, \top, (OS \setminus \{os\}) \cup \{\langle o, \text{inactive} \rangle\}, t \rangle} \quad (\text{Inactive})$$

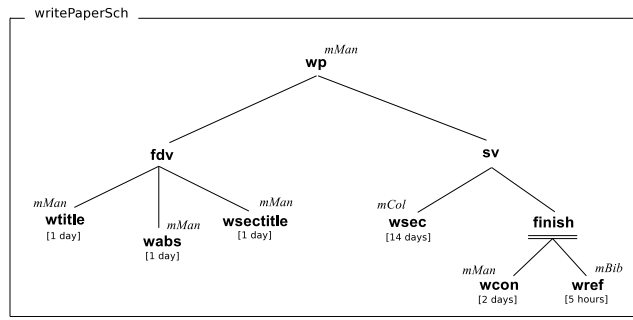
III. MOISE ORGANISATIONAL MODELLING LANGUAGE

MOISE proposes an organisational modelling language (OML) that explicitly decomposes the specification of organisation into structural, functional, and normative dimensions [4]. The structural dimension specifies the *roles*, *groups*, and *links* of the organisation. The definition of roles states that when an agent chooses to play some role in a group, it is accepting some behavioural constraints and rights related to this role. The functional dimension specifies how the *global collective goals* should be achieved, i.e. how these goals are decomposed (within *global plans*), grouped in coherent sets (through *missions*) to be distributed among the agents. The decomposition of global goals results in a goal-tree, called *scheme*, where the leaf-goals can be achieved individually by the agents. The normative dimension is added in order to bind the structural dimension with the functional one by means of the specification of the roles' *permissions* and *obligations* within missions.

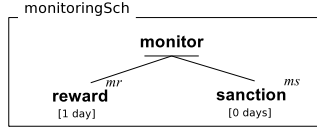
As an illustrative and simple example of an organisation specified using MOISE⁺, we consider agents that aim at writing a paper and therefore have an organisational specification to help them collaborate. Due to lack of space, we will focus on the functional and normative dimensions in the remainder of this paper. For the structure of the organisation, it is enough to know that there is only one group (`wpgroup`) where two roles (*editor* and *writer*) can be played.

To coordinate the achievement of the goal of writing a paper, a scheme is defined in the functional specification of the organisation (Fig. 3(a)). In this scheme, a draft version of the paper has to be written first (identified by the goal *fdv* in Fig. 3(a)). This goal is decomposed into three sub-goals: write a title, an abstract, and the section titles; the sub-goals have to be achieved in this very sequence. Other goals, such as *finish*, have sub-goals that can be achieved in parallel. The specification also includes a "time-to-fulfil" (TTF) attribute for goals indicating how much time an agent has to achieve the goal. The goals of this scheme are distributed in three missions which have specific cardinalities (see Fig. 3(c)): the mission *mMan* is for the general management of the process (one and only one agent must commit to it), mission *mCol* is for the collaboration in writing the paper's content (from one to five agents can commit to it), and mission *mBib* is for gathering the references for the paper (one and only one agent must commit to it). A mission defines all goals an agent commits to when participating in the execution of a scheme; for example, a commitment to mission *mMan* is effectively a commitment to achieve four goals of the scheme. Goals without an assigned mission are satisfied by the achievement of their sub-goals.

The normative specification relates roles to missions (Table I). For example, norm *n2* states that any agent playing the role *writer* has one day to commit to mission *mCol*. Designers can also define their own application-dependent conditions (as



(a) Paper Writing Scheme



(b) Monitoring Scheme

mission	cardinality
<i>mMan</i>	1..1
<i>mCol</i>	1..5
<i>mBib</i>	1..1

<i>mr</i>	1..1
<i>ms</i>	1..1

(c) Mission Cardinalities

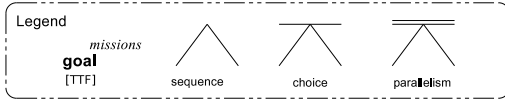


Fig. 3. Functional Specification for the Paper Writing Example

id	condition	role	type	mission	TTF
n1		editor	per	<i>mMan</i>	–
n2		writer	obl	<i>mCol</i>	1 day
n3		writer	obl	<i>mBib</i>	1 day
n4	unfulfilled(n2)	editor	obl	<i>ms</i>	3 hours
n5	fulfilled(n3)	editor	obl	<i>mr</i>	3 hours
n6	#mc	editor	obl	<i>ms</i>	1 hour

#mc stands for the condition “more agents committed to a mission than permitted by the mission cardinality”

 TABLE I
 NORMATIVE SPECIFICATION FOR THE PAPER WRITING EXAMPLE

in norms n4–n6). Norms n4 and n5 define sanction and reward strategies for fulfilment and unfulfilment of norms n2 and n3 respectively. Norm n5 can be read as “the agent playing role ‘editor’ has 3 hours to commit to mission *mr* when norm n3 is fulfilled”. Once committed to mission *mr*, the editor has to achieve the goal *reward*. Note that a norm in *MOISE* is always an obligation or permission to commit to a mission. Goals are therefore indirectly linked to roles since a mission is a set of goals.

IV. NORMATIVE ORGANISATION PROGRAMMING LANGUAGE

The NOPL is a particular class of NPL programs applied to *MOISE*. The syntax and semantics are the same as presented in Sec. II, but the set of facts, rules, and norms are specific for *MOISE* model and the organisational artifacts presented

in Sec. V. The main idea is that an OS is translated to different programs in NOPL, such programs define then the management of norms for groups and schemes. In this section we consider only the programs for schemes.

A. Facts

For scheme programs, the following facts, defined in the OS, are considered:

- $\text{scheme_mission}(m, \text{min}, \text{max})$: is a fact that defines the cardinality of a mission (e.g. $\text{scheme_mission}(\text{mCol}, 1, 5)$).
- $\text{goal}(m, g, \text{pre-cond}, \text{tff})$: is a fact that defines the arguments for a goal g : its mission, pre-conditions, and TTF (e.g. $\text{goal}(\text{mMan}, \text{wsec}, [\text{wcon}], \text{'2 days'})$).

The NOPL also defines some dynamic facts that represent the current state of the organisation and will be provided by the artifact that manage the scheme instance:

- $\text{plays}(a, \rho, gr)$: agent a plays the role ρ in the group instance identified by gr .
- $\text{responsible}(gr, s)$: the group instance gr is responsible for the missions of scheme instance s .
- $\text{committed}(a, m, s)$: agent a is committed to mission m in scheme s .
- $\text{achieved}(s, g, a)$: goal g in scheme s has been achieved by agent a .

B. Rules

Besides facts, we define some rules that will be useful for the norms. The rules are used to infer the state of the scheme (e.g. whether it is well-formed) and goals (e.g. whether it is ready to be achieved or not). Note that the semantics of *well-formed* and *ready goal* are formally given by these rules. As an example, some of such rules for the paper writing scheme are listed below.

```
// number of players of a mission M in scheme S
mplayers(M, S, V) :- .count(committed(_, M, S), V).

// status of a scheme S
well_formed(S) :-
  mplayers(mBib, S, V1) & V1 >= 1 & V1 <= 1 &
  mplayers(mCol, S, V2) & V2 >= 1 & V2 <= 5 &
  mplayers(mMan, S, V3) & V3 >= 1 & V3 <= 1.

// ready goals: all pre-conditions have been achieved
ready(S, G) :-
  goal(_, G, PCG, _) & all_achieved(S, PCG).
all_achieved(_, []).
all_achieved(S, [G|T]) :-
  achieved(S, G, _) & all_achieved(S, T).
```

C. Norms

We have three classes of norms in NOPL: norms for goals, norms for properties, and domain norms (which are explicitly stated in the normative specification). For the former class, we have the following norm:

```
// agents are obliged to fulfil their ready goals
norm ngoa:
  committed(A, M, S) & goal(M, G, _, D) &
  well_formed(S) & ready(S, G)
-> obligation(A, ngoa, achieved(S, G, A), 'now' + D).
```

This norm can be read as “when an agent A: (1) is committed to a mission M that (2) includes a goal G, and (3) the mission’s

scheme is well-formed, and (4) the goal is ready, then agent A is obliged to achieve the goal G before the deadline for the goal". This norm thus gives a precise semantics for commitment. It also illustrates the advantage of using a translation to implement the OMI instead of an object oriented programming language. For example, if some application or experiment requires a semantics of commitment where the agent is obliged to achieve the goal even if the scheme is not well-formed, it is simply a matter of changing the translation to a norm that does not include the `well_formed(S)` predicate in the activation condition of the norm. One could even conceive an application using schemes being managed by different NOPL programs (i.e. schemes translated differently).

For the second class of norms, only the mission cardinality property is considered in this paper since other properties are handled in a similar way. In the case of mission cardinality, the norm has to define the consequences of a circumstance where there are more agents committed to a mission than permitted in the scheme specification. As presented in Sec. II, two kinds of consequences are possible, obligation and regimentation, and the designer chooses one or the other when writing the OS. Regimentation is the default consequence and it is used when there is no norm with condition `#mc` in the normative specification. Otherwise, as in norm n6 of Table I, the consequence will be an obligation. The norm for mission cardinality regimentation is:

```
// norm for the cardinality regimentation
norm mission_cardinality:
  scheme_mission(M,_,MMax) &
  mplayers(M,S,MP) & MP > MMax
-> fail(mission_cardinality).
```

and the norm without regimentation is:

```
// norm for the cardinality regimentation
norm mission_cardinality:
  scheme_mission(M,_,MMax) &
  mplayers(M,S,MP) & MP > MMax
  responsible(Gr,S) & plays(A,editor,Gr)
-> obligation(A,mision_cardinality,
  committed(A,ms,_), 'now'+ '1 hour').
```

where the agent playing editor is obliged to commit to the mission `ms` in one hour.

For the third class of norms, each norm in the normative specification of the OML has a corresponding norm in NOPL. Whereas OML obligations refer to roles and missions, NPL requires that obligations are for agents and towards a goal. The NOPL norm thus identifies the agents playing the role in groups responsible for the scheme and, if the number of players still does not reach the maximum, the agent is obliged to achieve a state where it is committed to the mission. For example, the NOPL norm for norm n2 of Table I is:

```
norm n2:
  plays(A,writer,Gr) & responsible(Gr,S) &
  mplayers(mCol,S,V) & V < 5
-> obligation(A,n2,committed(A,mCol,S), 'now'+ '1 day').
```

V. ARTIFACT-BASED ARCHITECTURE

The proposals of this paper have been implemented on an OMI that follows the Agent & Artifact [14], [15]. In this approach, a set of organisational artifacts is available in the MAS environment providing operations for the agents so that they can interact with the OMI. For example, each

scheme instance is managed by an artifact. We can effortlessly distribute the OMI by deploying as many artifacts as necessary for the application.

Each organisational artifact has an NPL interpreter loaded with (i) the NOPL program automatically generated from the OS for the type of the artifact and (ii) dynamic facts representing the current state of (part of) the organisation. The interpreter is then used to compute: (i) whether some operation will bring the organisation into a inconsistent state (where inconsistency is defined by the designer by means of regimentations), and (ii) the current state of the obligations. The following algorithm, implemented on top of `CARTAGO` [16], shows the general pattern we used to implement every operation (e.g. role adoption, commitment to mission) in the organisational artifacts. In this new approach, the artifacts still provide an interface for the agents, and are now mostly programmed in NOPL instead of Java.

```
// let oe be the current state of the organisation managed by the
  artifact
// let p be the current NOPL program
// let npi be the NPL interpreter
when an operation o is triggered by agent a do
  oe' ← oe // creates a "backup" of current oe
  executes operation o to change oe
  f ← a list of predicates representing oe
  r ← npi(p, f) // runs the interpreter for the new state
  if r = fail then
    oe ← oe' // restore the state backup
    fail operation o
  else
    update obligations in the observable properties
    succeed operation o
```

We also developed a program that automatically generate the NOPL given an OS, however, due the lack of space, it is not presented here. The reader will find more details about this architecture, the translation, and a complete implementation of this OMI at https://sourceforge.net/scm/?type=svn&group_id=163721.

VI. RELATED WORKS

This work is based on several approaches to organisation, institutions, and norms (cited throughout the paper). In this section, we briefly relate and compare our main contributions to such work.

The first contribution of the paper, the NPL, should be considered specially for two properties of the language: its simplicity and its formalisation (that led to an available implementation). Similar work has been done by Tinnemeier et al. [7], where the operational semantics for a normative language was also proposed. They assume the availability of a snapshot of the global state of the organisation to evaluate activation of norms, which may hinder the implementation in a distributed scenario. Our NPL also requires a snapshot of

the organisational artifact state to evaluate norms, however the distribution problem is solved by generating different normative programs for several distributed artifacts where only the local state of the organisation is required. Another important difference is that in our approach the designer specifies the organisation in a higher-level language (OML) that is translated into a normative programming language (NOPL).

Regarding the second contribution, namely the automatic translation, we were inspired by work on ISLANDER [8], [17]. The main difference here is the initial and target languages. While they translate a normative specification into a rule-based language, we start from an organisational language and target at a normative language. It is simpler to translate organisational norms into NPL norms, since we have norms in both sides of the translation it is a 1-to-1 translation, than translate organisational norms into rules.

Regarding the third contribution, the OMI, we started from ORA4MAS [15]. The advantages of the approach presented here are twofold: (i) it is easier to change the translation than the Java implementation of the OMI; and (ii) from the operational semantics of NPL and the formal translation we are taking significant steps towards a formal semantics for MOISE.

VII. CONCLUSION

In this paper we showed an approach for translating an organisation specification written in MOISE OML into a normative program that can be interpreted by an artifact based OMI. Focusing on the translation instead of Java coding, we have brought flexibility to the development of the OMI. We also stressed the point that such a normative language can be based on only two basic concepts: regimentation and obligation. Prohibitions are considered either as regimentation or as an obligation for someone else. The resulting NPL is thus simpler to formalise (only 6 rules in operational semantics) and implement. Future work will explore NPL translations for other organisational and institutional languages. We also plan to prove correctness of the translation from OML into NOPL in future work.

REFERENCES

- [1] O. Boissier, J. F. Hübner, and J. S. Sichman, "Organization oriented programming from closed to open organizations," in *Engineering Societies in the Agents World VII (ESAW 06)*, ser. LNCS, G. O'Hare, M. O'Grady, O. Dikenelli, and A. Ricci, Eds., vol. 4457. Springer, 2007, pp. 86–105.
- [2] M. Esteva, D. de la Cruz, and C. Sierra, "ISLANDER: an electronic institutions editor," in *Proc. of the First International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2002)*, ser. LNAI 1191, C. Castelfranchi and W. L. Johnson, Eds. Springer, 2002, pp. 1045–1052.
- [3] V. Dignum, J. Vazquez-Salceda, and F. Dignum, "OMNI: Introducing social structure, norms and ontologies into agent organizations," in *Proc. of the Programming Multi-Agent Systems (ProMAS 2004)*, ser. LNAI 3346, R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah-Seghrouchni, Eds. Berlin: Springer, 2004.
- [4] J. F. Hübner, J. S. Sichman, and O. Boissier, "Developing organised multi-agent systems using the MOISE+ model: Programming issues at the system and agent levels," *International Journal of Agent-Oriented Software Engineering*, vol. 1, no. 3/4, pp. 370–395, 2007.

- [5] J. Ferber and O. Gutknecht, "A meta-model for the analysis and design of organizations in multi-agents systems," in *Proc. of the 3rd International Conference on Multi-Agent Systems (ICMAS'98)*, Y. Demazeau, Ed. IEEE Press, 1998, pp. 128–135.
- [6] D. V. Pynadath and M. Tambe, "An automated teamwork infrastructure for heterogeneous software agents and humans," *Autonomous Agents and Multi-Agent Systems*, vol. 7, no. 1-2, pp. 71–100, 2003.
- [7] N. Tinnemeier, M. Dastani, and J.-J. Meyer, "Roles and norms for programming agent organizations," in *Proc. of AAMAS 09*, J. Sichman, K. Decker, C. Sierra, and C. Castelfranchi, Eds., 2009.
- [8] A. García-Camino, J. A. Rodríguez-Aguilar, C. Sierra, and W. Vasconcelos, "Constraining rule-based programming norms for electronic institutions," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 18, no. 1, pp. 186–217, Feb 2009.
- [9] F. L. y López, M. Luck, and M. d'Inverno, "Constraining autonomy through norms," in *Proc. of first ICMAS*. ACM, 2002, pp. 674 – 681.
- [10] D. Grossi, H. Aldewered, and F. Dignum, "Ubi Lex, Ibi Poena: Designing norm enforcement in e-institutions," in *Coordination, Organizations, Institutions, and Norms in Agent Systems II*, ser. LNAI, P. Noriega, J. Vázquez-Salceda, G. Boella, O. Boissier, V. Dignum, N. Fornara, and E. Matson, Eds., vol. 4386. Springer, 2007, pp. 101–114, revised Selected Papers.
- [11] N. Fornara and M. Colombetti, "Specifying and enforcing norms in artificial institutions," in *Proc. of the 4th European Workshop on Multi-Agent Systems (EUMAS 06)*, A. Omicini, B. Dunin-Keplicz, and J. Padget, Eds., 2006.
- [12] J. Vázquez-Salceda, H. Aldewered, and F. Dignum, "Norms in multi-agent systems: some implementation guidelines," in *Proc. of the Second European Workshop on Multi-Agent Systems (EUMAS 2004)*, 2004.
- [13] G. D. Plotkin, "A structural approach to operational semantics," Computer Science Department, Aarhus University, Aarhus, Denmark, Tech. Rep., 1981.
- [14] A. Omicini, A. Ricci, and M. Viroli, "Artifacts in the A&A meta-model for multi-agent systems," *Autonomous Agents and Multi-Agent Systems*, vol. 17 (3), pp. 432–456, Dec. 2008.
- [15] J. F. Hübner, O. Boissier, R. Kitio, and A. Ricci, "Instrumenting multi-agent organisations with organisational artifacts and agents: "giving the organisational power back to the agents"," *Journal of Autonomous Agents and Multi-Agent Systems*, 2009.
- [16] A. Ricci, M. Piumi, M. Viroli, and A. Omicini, "Environment programming in CArtaGo," in *Multi-Agent Programming: Languages, Tools and Applications*, R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, Eds. Springer, 2009, ch. 8, pp. 259–288.
- [17] V. T. da Silva, "From the specification to the implementation of norms: an automatic approach to generate rules from norm to govern the behaviour of agents," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 17, no. 1, pp. 113–155, Aug 2008.

Towards an architecture for self-regulating agents: a case study in international trade

Brigitte Burgemeestre, Joris Hulstijn, and Yao-Hua Tan

Abstract—Norm-enforcement models applied in human societies may serve as an inspiration for the design of multi-agent systems. Models for norm-enforcement in multi-agent systems often focus either on the intra- or inter-agent level. We propose a combined approach to identify objectives for an architecture for self-regulating agents. In this paper we assess how changes on the inter-agent level affect the intra-agent level and how a generic BDI architecture IRMA can be adapted for self-regulation. The approach is validated with a case study of AEO certification, a European wide customs initiative to secure the supply chain while facilitating international trade.

Index Terms—self-regulation, agent architectures, norm compliance

I. INTRODUCTION

To motivate autonomous agents to comply with norms various enforcement mechanisms have been proposed. Norms here define standards of behavior that are acceptable in a society, indicating desirable behaviors that should be carried out, as well as undesirable behaviors that should be avoided [8]. Enforcement mechanisms often require the introduction of special “observers” or “regulator agents” that actively monitor the behavior of the other agents [1]. Such agents are assigned to monitor the behavior of other agents and sanction them in case of norm violations. When developing norm enforcement mechanisms for multi-agent systems, the modeling is often focused on the inter-agent level (between agents). Such models aim to analyze agent interactions and dependencies to construct norm enforcement mechanisms. The intra-level (inside the agent) is mainly treated as a black box. We argue that the intra- and inter-agent aspects cannot be viewed separately from each other, especially in norm enforcement where perceptions of external stimuli should motivate an agent to adapt its behavior and thereby its internal mechanisms.

Norm-enforcement models applied in human societies may serve as an inspiration for the design of electronic institutions

and open agent systems. An enforcement mechanism that elaborates on an agent’s internal architecture to achieve compliant behavior, and does not require additional ‘observers’ is self-regulation. Self-regulation is a control approach in which rule making and/ or enforcement are carried out by the agent itself, instead of a regulator agent or institution. It can be an alternative or extension to direct control, when external supervision and norm enforcement are not possible at all, are ineffective or when there is a lack of controlling resources. For example, in e-institutions it might be impossible to check all agent actions for compliance in real time. A solution then might be to do a code review up forehand and determine if an agent is compliant by design. In human societies programs of self-regulation have been found to contribute to expanded control coverage and greater inspectorial depth [2]. Self regulation can be implemented in various ways: from voluntary self regulation, where a group of agents voluntary chooses to regulate themselves, to mandated or enforced self-regulation, where a government agency delegates some of its regulative and enforcing tasks to the agents subjected to the norm, but retains the supervision, to a combination of mandated self regulation and direct control by regulator agents [10]. Each model of self-regulation causes different agent dependencies and information needs, which imposes different requirements on the IT architecture.

A special case of self regulation for international trade is the Authorized Economic Operator (AEO) program [7]. The AEO program is a European wide customs initiative that aims to secure the supply chain while at the same time reducing the administrative burden for companies through the use of self-control. Companies that are reliable in the context of customs related operations and have a good internal control system may apply for the AEO certificate and receive operational benefits from simplified customs procedures, preferential treatment, and less physical inspections. Companies that do not have an AEO certificate remain subject to the current level of customs controls. Participation in the AEO program is voluntary, but effective self-control is an obligatory requirement.

Implementing self-regulation as a control mechanism thus results in a redistribution or delegation of control tasks among the actors. Agents have to adapt their internal mechanisms to cope with these tasks. We see that changes at the inter-agent level affect the intra-level. We therefore propose a combined approach to develop an architecture to embed self-regulation as a control mechanism for multi-agent systems.

Manuscript received June 26, 2009. This research was partially funded by PGS IT Audit of the VU Amsterdam and the integrated project ITAIDE of the 6th Framework of the IST Programme of the European Commission

All authors are with the faculty of Economics and Business Administration of the VU University, Amsterdam, The Netherlands. The third author is also with the Dept. of Technology, Policy and Management of the Technical University Delft.

Brigitte Burgemeestre (e-mail: cburgemeestre@feweb.vu.nl).

Joris Hulstijn (e-mail: jhulstijn@feweb.vu.nl).

Yao-Hua Tan (e-mail: ytan@feweb.vu.nl).

In this paper we present our first steps towards an architecture for self-regulating agents. The research questions we like to answer in this paper are: 1. What objectives need to be met by an architecture on self-regulating agents? 2. How do we need to adapt existing Beliefs Desires Intentions (BDI) [9] architectures? As a starting point we propose a combination of frameworks to cover the inter- as well as the intra-agent analysis. For the inter-agent analysis the Intelligent Resource-Bounded Machine Architecture (IRMA) [3] is a good starting point because it is a general BDI architecture that is well accepted and has formed the basis for more recent agent architectures. Software engineering methodology TROPOS [6] provides suitable concepts to analyze and model agents' dependencies. We analyze direct regulation and self-regulation using TROPOS (Section II). Using this analysis we generalize the objectives for the internal architecture of a self-regulating agent. We try to embed the normative objectives in IRMA (Section III). Using the extended architecture and TROPOS model, we analyze a case study of AEO (Section IV). We examine if our adapted version of the architecture covers the findings of the case study. We identify its suitability and the shortcomings.

II. INTER-AGENT ANALYSIS

We first analyze the agents and the dependencies among agents. To do this we use concepts from the early requirements phase of the TROPOS methodology [4], which is derived from the i*conceptual framework[11]. The key concepts we use are: actor, goal, plan, resource and dependency. An actor can be an autonomous agent that has a goal or strategic interest. A goal can be satisfied through the execution of a plan, which is an abstract representation of a way of doing something. A resource can be a physical or informational entity. Actors can depend on each other to reach a certain goal, to execute a plan or to obtain resources. The agent that depends on another agent is called the depender, the agent he depends on is called the dependee. The object which is the subject of the dependency relation is called the dependum.

We first model the direct control approach where the actions of autonomous agents are regulated by special regulator agents. After that we analyze self-regulation and assess what changes when an autonomous agent internalizes control tasks of the regulator agent.

A. Agents' dependencies in direct control

In direct control we have two types of agents: an *Actor* agent (A) that is carrying out an activity and a *Regulator* agent (R) that is responsible for regulating A's actions such that agent A complies with the norms that are applicable to A. An agent can violate the norms through pursuing an illegal goal or by performing an illegitimate action. We assume that R has a norm framework from which it derives the set of norms tailored to an agent's specific situation. To regulate A, agent R has to have the following plans: R1: Specify norms for actor, R2 'Determine control indicators of actor', R3 'Monitor actor's actions' and R4 'Sanction actor'. R1 generates a set of

norms for A. R uses information about A and A's actions to select the appropriate norms from the norm framework that apply to A's specific situation. R2 determines 'control indicators' of A. A 'control indicator' is the kind of evidence required to demonstrate compliance of a norm, as well as infrastructural requirements to collect that evidence. For example: when a company sends an invoice, they always make a copy of the invoice and store the copy to be able to check if the invoice payments are correct and complete. R3 is the monitoring performed by R on A's actions, based on information provided by A about the control indicators. R4 describes the plan of R to sanction A in case of a norm violation. Agent A's model is quite simple, as A is a 'blind' agent that has no knowledge about the norms or control indicators and only acts. Therefore it is possible that A unknowingly engages in an activity that violates a norm that is imposed upon A by R. However, we do assume that A remembers action-sanction relations and that it can decide to cancel an action that will lead to a sanction. Figure 1 shows the dependency analysis for direct control.

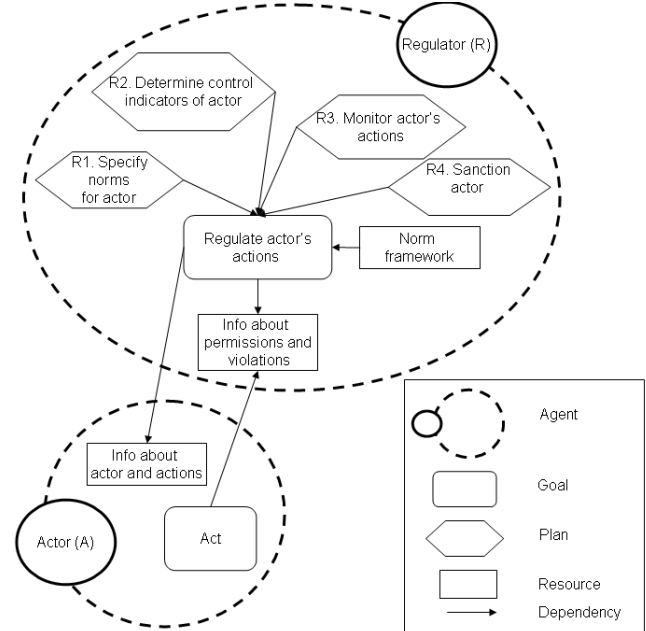


Figure 1: TROPOS model of direct control. The actions of an actor (A) are regulated by a regulator (R). Note that arrows depict dependency, not information flow. So to regulate A's actions, R depends on A for info about actor and actions.

B. Agents' dependencies in self-regulation

For self-regulation we start again with two types of agents: the actor agent (A) and the regulator agent (R). In self-regulation control tasks are delegated from R to A. Since A is autonomous, R can never be absolutely certain that A complies. R thus has to implement a mechanism to motivate A to regulate itself appropriately. Furthermore to maintain the power of the regulator to handle non-compliant agents, the sanctioning task (R4) remains the regulators responsibility.

We first consider the consequences of the internalization of

control tasks by A. Plans R1, R2 and R3 may be internalized by agent A as plans: A1 'Specify norms', A2 'Determine control indicators' and A3 'Monitor actions'. A1 specifies norms based on a norm framework which originates from R. This entails a new dependency between A and R: A now depends on R for communicating the norm framework. When the norm specification is done by A, A is also supposed to be able to differentiate between norm violations and norm compliance. A therefore no longer depends for information about violations and permissions on R, but has to do it himself. A2 defines control indicators about A's actions, based on the norms defined in A1. A3 describes the monitoring actions of A which it performs in the context of the control indicators from plan A2. The plans A1, A2, and A3 together, should support A to act compliantly with the norms. The acts of A in return affect the nature of the control actions. If A starts doing different activities the control indicators may become less effective and A therefore has to determine new control indicators that cover the norms. For example, if A replaces the process of sending paper invoices to its customers by sending them electronic invoices, new control indicators are required; e.g. log files instead of paper copies of the invoice.

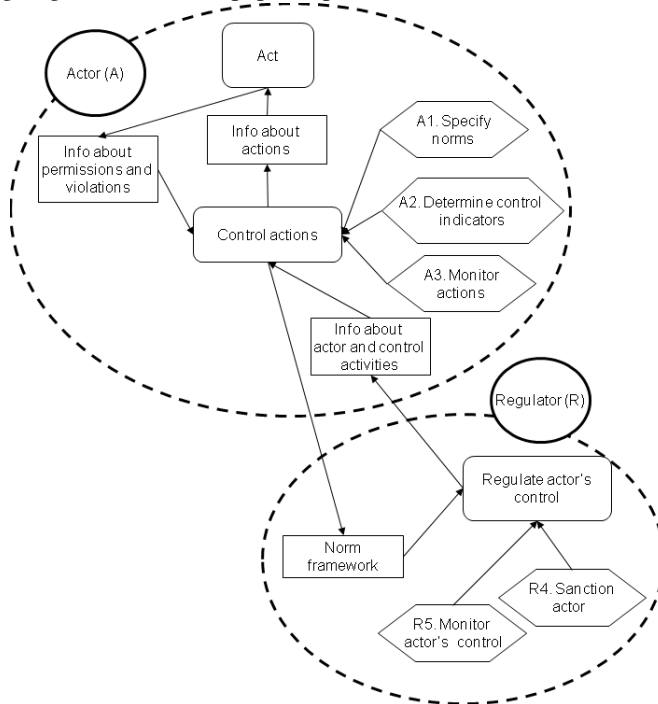


Figure 2: TROPOS model of self-regulation, control tasks of the regulator are internalized by the actor agent.

Now we describe the consequences of A's internalization of the control tasks of R's goals and plans. Since A now has to control its own actions, the goal of R to regulate A's actions is supposed to be met by the control activities of A. To determine if this delegation of control is effective, R's has adopted a new goal which is to regulate the control activities of A. To reach this goal, R also has defined a new plan (R5). R5 describes the activities of R to monitor and evaluate A's control actions. R now depends on A for information about its control activities

instead of its activities. In auditing R5 refers to a system-based audit, where the focus is on the control system itself instead of the business transactions. Before an agent thus can enter in a self-regulative relation it has to provide for its authenticated control architecture or control script to the regulator. Figure 2 shows the dependencies between agents A and R when they engage in self-regulation. When we compare direct control with self-regulation we see that A internalizes some of R's control activities on A. New information resources have to be gathered to be used within the control activities. Also new goals evolve and consequently the adoption of new plans. In correspondence new dependencies between R and A develop for the acquisition of other information resources

Summarizing, a self-regulating agent has to have the capabilities to: (1) Detect, internalize and store applicable norms in the environment, (2) Translate norms into measurable control indicators, and (3) 'Monitor, detect and mitigate possible norm violations'. In the next section we zoom into the internal architecture of the actor agent in self-regulation

III. INTER-AGENT ANALYSIS

We now analyze how the new tasks and dependencies revealed by the TROPOS models affect an agent's internal architecture. We acknowledge that these tasks are complex normative tasks. As a basis for our model we use the Intelligent Resource-Bounded Machine Architecture (IRMA) [3]. The architecture is a BDI architecture where the intentions are structured into plans. A plan can be the plan that an agent has actually adopted, or a plan-as-recipe that is stored into the plan library. Plan options are proposed as a result of means-end reasoning or by the opportunity analyzer. The opportunity analyzer detects changes in the environment and determines new opportunities, based on the agent's desires. The options are filtered through a compatibility filter, that checks the options to determine compatibility with the agent's existing plans, and a filter override mechanism, in which the conditions are defined under which (portions) of plans need to be suspended and replaced by another option. The deliberation process determines the best option on the basis of current beliefs and desires.

Consider an autonomous agent that likes to achieve a certain goal. The agent has already several plans of action available (in its plan-library) to reach this goal. Before deliberating on a plan, the agent engages in a filtering process. This process constrains the agent's possible plans, to plans that can be completed given its available (sub) plans in the plan library, its beliefs and desires. The agent chooses from this selection the best plan, given its beliefs and desires, and executes the plan. Figure 3 shows our extension of the IRMA architecture, adapted for self-regulation. Norm related adaptations are shown in grey and dotted lines. The ovals in the figure are information stores (repositories) and the rectangles are process modules.

Within IRMA we like to implement the processes and information stores that are needed for self-regulation. A self-

regulating agent needs to internalize certain control activities to control its actions. The activities are: specify norms (A1), determine control indicators (A2), and monitor actions (A3). These control activities require input from the agent's actions, and the actions in turn are influenced by the norms. We first analyze what modules IRMA are possibly affected by normative reasoning

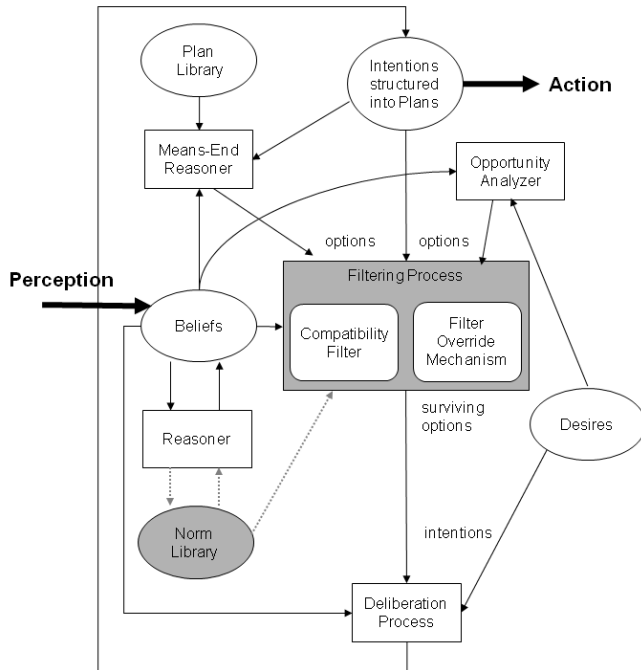


Figure 3: A reasoning component for self-regulating agents adapted from [3]

Norms can impact the information stores and or processes of the architecture. A norm can be implemented in plans and function as a threshold to restrict the outcome. For example, a thermostat function that tries to keep the room heated at a certain temperature. Norms can also restrict the possible set of plans. Plans that violate the norm are not stored in the plan library. Or in means-end reasoning: there are illegal plans available in the plan library but we do not consider them as appropriate options to reach a goal. Norms can also prevent the actual execution of a plan. For example, a person can plan to rob a bank, but decide not to do so.

Besides that, norms affect the beliefs, and beliefs affect the norms. An agent may realize, based on its beliefs, that it is acting non-compliant with the norms. Or, an agent realizes that due to a change in activities certain norms are no longer applicable and new norms must be incorporated. When an agent adopts a new norm, this must be known (believed).

Norms are also related to the desires of an agent. An agent's desires may violate the norms. For example, an agent may desire a handbag that is made of the skin of a protected snake. A norm is that killing a protected animal is illegal. If norms are included in the compatibility filter, an agent can check if an option is compatible with its norms. If norms are part of the filter override mechanism, non-compliance can be a condition under which an agent always has to reconsider its plans. Both

implementations make it possible for an agent to decide not to consider a plan option that aims at buying a snake skin handbag. The opportunity analyzer may use the norms and beliefs to search for an alternative, such as a fake snake skin handbag.

We find that norms can impact all components of the architecture. To assure consistent norm application we propose a central information-storage for norms similar to what the plan library is for plans. Activity A1 updates the norm library according to the beliefs of the agent. Only norms that are considered to be applicable to the agent's specific situation are included. To make an agent aware of a norm (violation) we connect the norm library with the reasoner module that is attached to the beliefs. If an agent then reasons about its beliefs, it takes the norms into account. Beliefs about a norm (violation) can be used as input for the means-end reassembler, opportunity analyzer and the deliberation process. Besides that, the agent may use its knowledge about norms to determine the control indicators of A2. We consider the filtering process the best location to implement the control indicators. Beliefs about norms are already included in the other reasoning processes. The filtering process and reasoning thus together consider (non-) compliant behavior. We think that the majority of the control indicators should be embedded in the compatibility filter and only severe violations should be handled by the filter override mechanism. Otherwise it could happen that the filtering is too strict. The monitoring in A3 is handled through a comparison of the beliefs about the data on the indicators with the norms. Based on results from this analysis controls in the filtering process may be adapted. Figure 3 shows an adapted version of the rational agent reasoning architecture for self-regulation.

Our approach of embedding norms into the filtering process is compatible with the framework that is proposed by [8]. Norms can also be implemented into the goal generation mechanism as was done in the BOID architecture [5]. In BOID one can distinguish two kinds of goals: internal motivations (desires), representing individual wants or needs, and external motivations (obligations) to model social commitments and norms[5]. All these potential goals may conflict with each other. To resolve conflicts among the sets of beliefs, obligations, intentions and desires, a priority order is needed. In the BOID, such a (partial) ordering is provided by the agent type.

IV. CASE STUDY AEO CERTIFICATION

We use our models to analyze a specific case of self-regulation: AEO certification. The case study results are based on document analysis and a series of semi-structured interviews with experts from Dutch Tax and Customs Administration, held in the period of May till November 2009. Meeting notes were made by the authors and verified by interview partners. Intermediate results of the case study were validated in a one-day workshop.

An Authorized Economic Operator (AEO) can be defined as

a company that is in-control of its own business processes, and hence is reliable throughout the EU in the context of its customs related operations [7]. Typically, modern enterprise information systems (e.g ERP, CRM etc.) play an essential role for companies to be in-control. AEO's will receive several benefits in customs handling, such as a "Green Lane" treatment with a reduced number of inspections. These benefits can lead to considerable cost-reductions for businesses. For non-certified enterprises customs will continue to carry out the traditional supervision. Customs can thus direct their efforts towards non-certified companies to increase the security of international supply chains, while at the same time reducing the administrative burden for AEOs.

To qualify as AEO, a company must meet a number of criteria, which are described in the community customs code and the AEO guidelines [7], which are developed by the European Commission. Part of the application procedure is a self-assessment on the quality of the company's internal control system for aspects that are relevant to the type of AEO certificate ('Customs simplifications', 'Security and safety' or 'Combined' [7]). The company's approach and the results of the self-assessment are inspected by customs. The customs determine whether the self-assessment is performed well and whether the results indicate that a company is able to control its business processes such that they contribute to a secure supply chain. If this is the case and the other requirements are met an AEO certificate is issued by the customs office. Next we focus on the self-assessment task.

A. The self-assessment task

The company's first task is to collect information related to the specific nature of the company to focus the self-assessment. This step is called 'Understanding the business'. The next step is to identify (potential) risks to which the business is exposed using the AEO guidelines, which provide an overview of general risk and attention points. The company determines which sections are important according to the nature of the business activities. A company then has to identify, what risks affect the supply chain's safety, and are therefore of interest of the customs authorities. The company thus replaces the customs' task of risk identification. For example, computer components are valuable goods, which are subject to theft. Trading valuable goods requires more security measures, than, say, trading in a mass product like fertilizer. However, some ingredients of fertilizer may be used to assemble explosives, leading to a different set of risks

A company then assess if appropriate internal control measures are taken to mitigate these risks. The vulnerability of a company to threats depends on its current control measures. Control measures either reduce the likelihood, by dealing with vulnerabilities (preventative controls), or reduce the impact (detective and corrective controls). A robust system of controls is thus able to prevent, detect and correct threats. A robust system of controls should also monitor its own functioning. For risks that are not controlled, additional measures may be implemented or the risk is "accepted". Risks can be accepted,

if the likelihood of a threat is limited and the risk is partially covered, or if the costs for complete coverage are very high.

The company has to motivate its choices in its system of control measures to customs. It has to show how its risk management approach contributes to being a self-controlling and reliable party. The company therefore evaluates the effective implementation of the proposed measures, using the COSO internal control scoring definitions. COSO is a framework for risk management and internal control [12]. The scores range from 0 "no control measures in place", 1 "internal control is ad hoc and unorganized", 2 "internal control has a structured approach", 3 "internal control is documented and known", 4 "internal control is subject to internal audits and evaluation" until 5 "internal control measures are integrated into the business processes and continuously evaluated". This scoring provides the customs with an indication of the maturity level of the company's self-controlling abilities.

B. Case analysis

In the AEO case study we see the implementations of tasks A1, A2, and A3 at the company's side. A company has to define a control system appropriate to handle its specific risks. The company therefore translates the general AEO guidelines into norms that are applicable in its own practice and circumstances (compatible with A1). Thereby a company determines parameters to control its business processes (A2). A company with a control system of a high maturity level monitors its actions (A3) through internal audits and controls that are integrated in the processes. The customs replaces its traditional controls of the company's processes (R1, R2, R3) by an assessment of the company's self-regulating capabilities and monitors the control actions of the company (R5). We also observe dependencies on information needs. The company depends on abstract norms (e.g. the AEO Guidelines) provided by the customs, which they try to apply to themselves as customs would do. The customs on the other hand depends on the company for information about their control system.

The AEO case provides us a new approach of control that could be applied to a multi agent system. It shows that norm enforcement can be a task that can be distributed between various types of agents. Furthermore we learned that self-regulation only works under certain conditions and that delegating control tasks is not simple. In general companies find it difficult to do a self-assessment as they do not know what customs expects from them. Especially the specification of abstract norms of the AEO guidelines into company specific concrete norms proved to be hard. For companies it is thus unclear when they have taken sufficient measures to secure their part of the supply chain. Companies expect from the customs to indicate on a more detailed level what is sufficient: "A fence for a chemical company should be X meters high". Even for customs such knowledge is often only implicitly available as "expert knowledge" that is difficult to externalize and make accessible for companies.

When we look at the company's internal control system we see that norms have to be internalized based on perceptions of

the environment. Only applicable norms are implemented. The norms have to be implemented in a systematic and structured way such that they detect norm violations and prevent them from occurring. In the architecture we see norms implemented as a filtering mechanism. In the AEO certification we see norm control as a structured process. In addition, mature self-controlling companies may have controls integrated in the processes or audits to check the functioning of the controls. The total control system of a company could be seen as their implementation of the internal control architecture. Therefore these new monitoring activities of customs in the AEO case could be seen as quality assessment of such a control architecture rather than the traditional role of Customs to control the specific business operations of the company. This fundamental change in the controlling role of the government is often referred to as the transformation from operational control to meta-control, where operational control is delegated by the Customs to the companies themselves.

V. DISCUSSION

The combination of TROPOS and IRMA for self-regulating agents also has its limitations. However, we do not claim that these are the best approaches currently available. Instead we used the approaches as a means to identify requirements for self-regulating agent at the intra- and inter-agent level. Below we describe the two most important limitations.

First, the most important limitation of the architecture is that it is not *reflective*. By this we mean that agents cannot learn from their mistakes. When the agent determines that a plan contains or leads to a norm violation it is only able to cancel this plan as a current possible option. It lacks mechanisms to delete or change such plans in a plan library. Desires that violate norms can also not be changed. The agent therefore keeps proposing violating plans and desires. Since norms are context dependent it is quite complex to differentiate violating plans from non-violating plans. Plans that are allowed in one situation may be a violation under different circumstances. An adaption of the plan mechanism is needed.

Secondly, there seems to be fundamental problem in delegation of control; namely that often it is not clear how to communicate the delegated norms from the regulator agent to the regulated agent. For companies it is difficult to interpret and implement the customs' norms for their business activities. Should customs and companies implement protocols, a vocabulary or procedures such that they effectively can communicate information? How should a company make its internal control system available to customs, such that they can determine the quality of a control system in a specific context with limited expert knowledge? These and related questions have to be answered through a study of norm communication between agents.

VI. CONCLUSION AND FURTHER RESEARCH

A combined approach, that analyses the inter- (between agents) and intra-agent level (inside agents), was suitable to

identify objectives for an architecture for self-regulation. We identified key processes and their influence on the dependencies between agents and the internal agent architecture. The models provide insight in differences in requirements for direct controlled agents and self-regulating agents. The analysis also points out the limitations of some well-known existing approaches. IRMA lacked in reflective capabilities and is therefore not sufficient to model a truly self-regulating agent: an agent that is able to learn from its experiences with norms and use these experiences as constraints for future normative reasoning. Also unaddressed were aspects of norm communication. For two agents to engage in a self-regulation relation, they must be able to communicate the norms effectively. Since the agents are autonomous we cannot simply assume that both agents use similar vocabularies or protocols [6]. A solution for norm communication should take the agent's autonomy into account.

Future research will zoom in on the role of reflection on normative behavior and the communication of norms. Besides that we are also interested in the evolution process of an agent from direct control to self-regulation.

Acknowledgments We would like to thank the Dutch Tax and Customs administration for their discussions.

REFERENCES

- [1] G. Boella, L. van der Torre, and H. Verhagen: Introduction to normative multiagent systems. *Computational and Mathematical Organization Theory*, 12:71–79, 2006P
- [2] J. Braithwaite: Enforced self-regulation: a new strategy for corporate crime control. *Michigan law review* vol. 80, pp 1466-1506, 1982
- [3] M. E. Bratman, D. Israel, and M. Pollack. Plans and resource-bounded practical reasoning. In R. Cummins and J. L. Pollock, editors, *Philosophy and AI: Essays at the Interface*, pages 1--22. The MIT Press, Cambridge, Massachusetts, 1991.
- [4] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems*, 8, pp. 203–236, 2004.
- [5] J. Broersen, M. Dastani, J. Hulstijn, Z. Huang and L. van der Torre. The BOID architecture - Conflicts between beliefs, Obligations, Intentions and Desires. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pp. 9-16, Montreal Canada, May 2001
- [6] C.B. Burgemeestre, J. Liu, J. Hulstijn and Y. Tan: Early Requirements Engineering for e-Customs Decision Support: Assessing Overlap in Mental Models. In the *Forum Proceedings of the 21st CAiSE conference*, pp 31-36, Amsterdam, The Netherlands 2009
- [7] European Commission: AEO Guidelines, TAXUD/2006/1450, 2007. http://ec.europa.eu/taxation_customs/customs/policy_issues/customs_security/aeo/
- [8] F. Meneguzzi and M. Luck: Norm-based behaviour modification in BDI agents. *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Budapest, Hungary, 2009.
- [9] A.S. Rao and M.P. Georgeff: Modelling rational agents within a BDI-architecture, in *proceedings of Knowledge representation and reasoning (KRR-91) Conference*, San Mateo CA, 1991.
- [10] J. Rees, Self Regulation: An Effective Alternative to Direct Regulation by OSHA?, *Policy Studies Journal*, 16:3 pp 602-614, 1988
- [11] E.K.S. Yu, Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering, in: *Proceedings of the Third IEEE International Symposium on Requirements engineering*, pp. 226-235, 1997
- [12] COSO enterprise risk management framework. Available: <http://www.coso.org>

Four Ways to Change Coalitions: Agents, Dependencies, Norms and Internal Dynamics

Guido Boella
University of Turin
Italy
Email: guido@di.unito.it

Leendert van der Torre
University of Luxembourg
Luxembourg
Email: leon.vandertorre@uni.lu

Serena Villata
University of Turin
Italy
Email: villata@di.unito.it

Abstract

We introduce a new formal approach to social networks in order to distinguish four ways in which coalitions change. First, the agents in the network change. Second, dependencies among the agents change, for example due to addition or removal of powers and goals of the agents. Third, norms can introduce normative dependencies for obligations and prohibitions. Fourth, coalitions can change due to internal processes. We propose a number of stability measures to identify each one of the four proposed sources of coalitions' dynamics and the consequences they induce on the stability of coalitions.

1. Introduction

Coalitions play a central role in social reasoning, and thus various theories have been used and developed in multiagent systems. For example, coalitional game theory has been adopted from economics and extended for multiagent systems [6], [7], and social networks have been adopted from social sciences and modified to represent dependence networks among agents [8], [4], [5]. These theories differ in various ways. For example, in the former, potential coalitions may be seen as sets of agents while in the latter, dependence networks can be seen as criteria for proposing/accepting to form coalitions [8], or *potential* coalitions are viewed as sets of dependencies (the dependencies represent the contract of the potential coalition) [5]. Moreover, in the former various notions of stability are defined, whereas in the latter they are not. In this paper, we address the question how to distinguish and model the different reasons behind the change of coalitions in requirements analysis.

Possible reasons behind these changes are due to operations of addition and removal of the components of our model such as agents, dependencies among agents, normative dependencies concerning normative goals and powers. More precisely, how do we measure the evolution and the changes of a coalition over time in terms of:

Changes of the agents and dependencies. We distinguish two kinds of uses for dependence networks: global use in

software engineering where the designer models all stakeholders [2], and social simulation where no such assumption is made [8]. In the former, game theory can be used for reasoning about social interaction, in the latter simulation methods are used. We follow the tradition of TROPOS [2] for requirements analysis, as formalized by Sauro [5] and close to qualitative game theories developed by Wooldridge et al. [1], not the latter [8].

Changes of the dependencies related to norms. Norms are used for the dynamics of dependence networks, which explained why they have not been considered thus far in the static dependence networks [9]. A norm analytically implies that agents (intend to) execute them, and therefore leads to dependencies among agents just like the original goal-based dependencies studied by Sichman and Conte [9]. Norms should be clearly distinguished from obligations. More precisely, norms are used to generate new dependence networks in which a number of dependencies are normative ones. Within a dependence network, the effect of the norm consists in a normative goal such as an obligation. These normative goals, i.e., obligations, are treated just like goals derived from the agent's desires. The coalitions which may emerge depend on the dependencies among the agents, so since norms change the dependencies among agents, they also change the coalitions which will emerge.

Internal dynamics. Changes of the coalition itself in terms of goal-based and norm-based dependencies composing the coalition, e.g., an agent is excluded from a coalition because of a malicious behaviour.

We call the last kind of change *internal dynamics* to distinguish it from the other dynamics related to the addition or deletion of agents or goal-based and norm-based dependencies. They represent the case in which the network remains the same, involving the same agents and dependencies, but the composition of the coalition changes, including new dependencies or excluding the old ones. A simple and intuitive common sense example of the above presented changes can be the next one. Consider a soccer team as a coalition. It can change because new players come in, or players retire. It can change, because agents acquire new abilities or loose abilities, e.g., they loose their form, they

break a leg, and so on, or get new goals, e.g., they want to play in the national team. Concerning norms, there can be the obligation set by the trainer for a player to play in the left wing position. Concerning internal dynamics, there may be a malicious behavior of a player, e.g., he gets too many red cards since he is too aggressive and he is no longer allowed to play. In the paper, we explain the changes using a grid-based running example.

From the multiagent systems field, we use the normative multiagent paradigm while from social network theory we take the idea of defining graph theoretic measures. Concerning measures, we define measures associated to the number of agents and the number of goal-based dependencies present in each time instant, counting the number of norm-based dependencies in each time instant and counting the changes in the dependencies composing coalitions. Our measures are unified in an average measure returning coalitions' stability depending on the differences between values associated to consecutive time instants.

In this paper, we do not give a formal ontology but we define indications of the possible changes of coalitions. Moreover, we do not perform any simulation as in Carley's dynamic networks analysis [3]. This paper is organized as follows. Section 2 presents a grid-based scenario. Section 3 and 4 present the key concepts of our metamodel and the three coalitions' changes in detail. Related work and conclusions end the paper.

2. Changing coalitions in a GRID scenario

We use the following example of a coalition in a grid environment. Inside a virtual organization (VO), local coalitions may be formed in order to cooperate to achieve shared goals such as, i.e., computations and storage of satellites' data. We depict a section of the VO composed by five nodes, as in Figure 1.a, following the legend of Figure 3. The VO is composed by four nodes connected to each other by dependencies based both on goals and on norms and nodes a , b and c form a local coalition. Considering goal-based dependencies, node b depends on node a to save the file *satellite.jpg*, node c depends on node b to save the file *satellite.mpeg* and node c depends on node d to run the file *results.mat*, since they are not able to perform their goals alone. Considering norm-based dependencies, instead, node a depends on node c to have the permission to open the file *dataJune.mat* while node c is obliged to give to node b the results of the running of file *mining.mat*.

The first kind of change of coalitions in the grid scenario follows directly from the grid metaphor. Computers can be connected to the grid like electrical machines can be connected to the power net. So the computers connected to the grid changes frequently, e.g., node e . If they do so, then also the coalition changes. How frequently they change is our first measure.

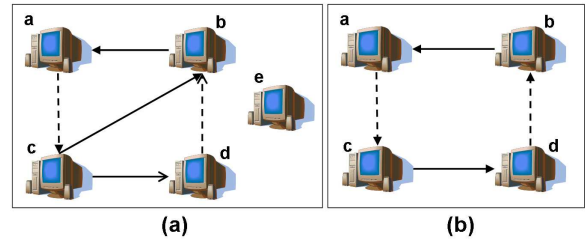


Figure 1. Grid network: $C=\{a, b, c\}$; $C=\{a, b, c, d\}$.

The second kind of change concerns goal-based dependencies. Node b fulfilled the goal of node c to save the file *satellite.mpeg*. This dependency does not hold anymore and it is deleted, as shown in Figure 1.b. This deletion of dependencies changes the structure of the local coalition because of now the reciprocity involves also node d inside the system. The deletion, as the addition, of a goal-based dependency may cause a change in the coalitions composed by these dependencies.

The third kind of change is related with security. A node has a number of private information, e.g., a unique access to its pc. If another node has the necessity to access to it, it has to ask the first node the permission, e.g., a login and a password, as in the norm-based dependency among nodes a and c . Obligations, instead, are due to particular services provided by the nodes. The obligation is represented as a dependency, as in the case of the norm-based dependency among nodes d and b , and it is removed if the obligation is no more active in the system. Figure 2.a shows the introduction of a norm-based dependency representing the obligation for node b to give the access to file *finalres.txt* to node a .

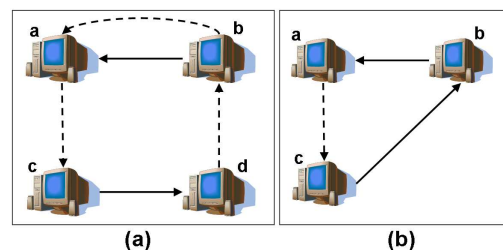


Figure 2. Grid network: $C=\{a, b, c, d\}$; $C=\{a, b, c\}$.

The fourth kind of change, internal changes of coalitions, represents changes in the composition of the coalition because of internal reasons. In Grid networks, malicious behaviors can be recognized, e.g., in case of attacks or for not properly following the protocol, and malicious nodes can be excluded from further interactions with the other nodes, as shown in Figure 2.b.

3. The model

3.1. The model definition

Our modeling approach aims to provide a design methodology both for multiagent systems and social systems, based on the normative multiagent paradigm. We present our model as a tuple composed by the concepts of agents, goals, norms and time. These notions are represented in our dependency modeling as nodes or dependency relations between these entities. For more details about the dependency modeling, see Villata [10]. Our model can be represented as follows:

Definition 1: $\langle A, G, N, T, D, D \subseteq A \times A \times G, T \rightarrow 2^A, T \rightarrow 2^D, N \rightarrow 2^D, C \subseteq 2^D, N \subseteq C \rangle$ consists in a set of agents A , a set of goals G , a set of norms N , a set of time instants T and a set of dependencies D . Every time instant is related to the set of agents and to the set of dependencies D present in the system in that instant. Norms are represented as a subset of dependencies. A coalition is represented as a set of dependencies and a subset of the dependencies composing a coalition can be represented by norms.

In this model, a coalition can be represented by a set of dependencies, represented by $C(a, B, G)$ where a is an agent, B is a set of agents and G is a set of goals. Intuitively, the coalition agrees that for each $C(a, B, G)$ part of the coalition, the set of agents B will see to the goal G of agent a . Otherwise, the set of agents B may be removed from the coalition or be sanctioned.

In a multiagent system, since an agent is put into a system that involves also other agents, he can be supported by the others to achieve his own goals if he is not able to do them alone. This leads to the concept of power representing the capability of a group of agents (possibly composed only by one agent) to achieve some goals (theirs or of other agents) performing some actions without the possibility to be obstructed. The power of a group of agents is defined as follows:

Definition 2 (Agents' power): $\langle A, G, power : 2^A \rightarrow 2^{2^G} \rangle$ where A is a set of agents, G is a set of goals. The function *power* relates with each set $S \subseteq A$ of agents the sets of goals G_S^1, \dots, G_S^m they can achieve.

Definitions 1 and 2 have the aim to explain how social dependence networks can be seen as multiagent systems. The notion of power is relevant for our methodology since it represents the social basis for the development of our model based on the methodology of dependence networks as developed by Conte and Sichman [9]. In this model, an agent is described by a set of prioritized goals, and there is a global dependence relation that explicates how an agent depends on other agents for fulfilling its goals. For example, $dep(\{a, b\}, \{c, d\}) = \{\{g_1, g_2\}, \{g_3\}\}$ expresses that the set of agents $\{a, b\}$ depends on the set of agents $\{c, d\}$ to see

to their goals $\{g_1, g_2\}$ or $\{g_3\}$. A dependence network is defined as follows:

Definition 3 (Dependence Networks (DN)): A dependence network is a tuple $\langle A, G, dep, \geq \rangle$ where:

- A is a set of agents and G is a set of goals;
- $dep : 2^A \times 2^A \rightarrow 2^{2^G}$ is a function that relates with each pair of sets of agents all the sets of goals on which the first depends on the second.
- $\geq : A \rightarrow 2^G \times 2^G$ is for each agent a total pre-order on goals which occur in his dependencies: $G_1 \geq (a)G_2$ implies that $\exists B, C \subseteq A$ such that $a \in B$ and $G_1, G_2 \in depend(B, C)$.

The *dependency modeling* represents our modeling activity consisting in the identification of the dependencies among the agents. Our *dependency modeling* is represented as a directed labeled graph whose nodes are instances of the concepts of the metamodel, e.g., agents, goals, and whose arcs are instances of the notions representing relationships between them such as goal-based dependency and norm-based dependency. A graphical representation of the model obtained following this modeling activity is depicted in the legend of Figure 3. Open and closed arrows are used to provide an immediate graphical representation of coalitions.

4. Coalitions' Dynamics

In this section, we present a definition of coalition based on the structure of dependence network and how to use these different kinds of dependencies to model and measure coalitions' dynamics. In our model, a coalition is defined as follows:

Definition 4 (Coalition): Let A be a set of agents and G be a set of goals. A coalition function is a partial function $C : A \times 2^A \times 2^G$ such that $\{a \mid C(a, B, G)\} = \{b \mid b \in B, C(a, B, G)\}$, the set of agents profiting from the coalition is the set of agents contributing to it. Let $\langle A, G, dep, \geq \rangle$ be a social dependence network, a coalition function C is a coalition if $\exists a \in A, B \subseteq A, G' \subseteq G$ such that $C(a, B, G')$ implies $G' \in dep(a, B)$.

As introduced before, we can model and measure coalitions' dynamics over time in terms of: changes of the agents and goal-based dependencies, changes of the dependencies related to norms and changes inside the coalition itself.

4.1. Agent and dependencies' changes

The first kind of change is due to agents entering or leaving the multiagent system we model or to the dependencies added or deleted depending on the fulfillment of the related goal or the presence of the power to fulfill this goal. In our model, we distinguish two different kinds of goals, achievement goals and maintenance goals. In contracts goals are typically achievement ones while, in game

theoretical approaches, coalitions are typically concerned with maintenance goals. In this paper, we assume that goals are maintenance goals rather than achievement ones, which give us automatically a longer term and a more dynamic perspective to define the evolution of coalitions and thus their stability. Moreover, our model aims to distinguish and represent not only short term situations such as, for example, a virtual meeting on Second Life but also long term situations as, for example, the work of a particular department or office or, in the Grid scenario, the work of a virtual organization for e-Research.

We can define two measures associated to the number of agents and the number of goal-based dependencies present in each time instant. The first measure calculates the ratio between the number of agents added and removed in a particular time instant depending and the number of agents present at the previous time instant. The second measure calculates the ratio between the number of goal-based dependencies added and deleted in a particular time instant depending and the number of goal-based dependencies present at the previous time instant. The measures are defined as follows:

Definition 5 (Agents and Dependencies Measures): Let i be a time frame, N_i^{Agent} is given by the number of agents entering the system A_i^+ and leaving the system A_i^- , depending on the total number of agents A_{i-1} present at time frame $i - 1$:

$$N_i^{Agent} = \sum \left(\frac{A_i^+}{A_{i-1}} \right) + \sum \left(\frac{A_i^-}{A_{i-1}} \right)$$

Let i be a time frame, N_i^{Dep} is given by the number of goal-based dependencies added to the network D_i^+ and deleted from the network D_i^- , depending on the total number of goal-based dependencies D_{i-1} present at time frame $i - 1$:

$$N_i^{Dep} = \sum \left(\frac{D_i^+}{D_{i-1}} \right) + \sum \left(\frac{D_i^-}{D_{i-1}} \right)$$

Example 1: In Figure 3, we present the case of six time frames visualizing the evolution of a coalition. In the first time frame, we have five agents and a coalition involving agents a, b, c , as shown by the dependencies composing it. There are also two norm-based dependencies and three goal-based dependencies. The passage from the first instant t_1 to the second one shows the deletion of agent e . From instant t_2 to instant t_3 , we observe the deletion of the goal-based dependency connecting agents c and b . Also the coalition changes and it is formed by all the four agents. From instant t_3 to instant t_4 , the situation changes back to the original configuration but the coalition is fixed. From instant t_4 to instant t_5 , agent d disappears, a norm-based dependency is deleted and the coalition changes its actors, involving now a, b and c . From instant t_5 to instant t_6 , the situation comes back to the situation of instant t_4 .

4.2. Norms' changes

The second kind of change is due to norms and, in particular, to obligations. An obligation is a requirement which must be fulfilled to take some course of action, whether legal or moral. Normative reasoning is strictly related to norms' changes and the definition of a representation and a measure for them allows to do it. The norm sets a particular kind of dependency among two agents. This dependency can be deleted if the obligation is fulfilled or a new obligation can be inserted into the system to regulate its behaviour. In our model, we distinguish, represent and measure both short term contracts, e.g., a transaction on e-Bay such as an agreement carried out between separate entities involving the exchange of items of value as goods and money, and long term contracts, e.g., the marriage contract which hopefully lasts forever.

We can define a measure associated to the number of norm-based dependencies present in each time instant. This measure calculates the ratio between the number of norm-based dependencies added and deleted to each time instant depending and the total number of norm-based dependencies present in that time instant. The measure is defined as follows:

Definition 6 (Norms Measure): Let i be a time frame, N_i^{Norm} is given by the number of norm-based dependencies added to the network O_i^+ and deleted from the network O_i^- , depending on the total number of norm-based dependencies O_{i-1} present at time frame $i - 1$:

$$N_i^{Norm} = \sum \left(\frac{O_i^+}{O_{i-1}} \right) + \sum \left(\frac{O_i^-}{O_{i-1}} \right)$$

Example 2: In Figure 4, we model three time instants. In the first time instant t_1 , we have a coalition formed by all the four agents, three goal-based dependencies and two norm-based dependencies. From time instant t_1 to time instant t_2 , the norm-based dependency involving agents d and b is removed due to the removal of the normative goal or the removal of the associated power. From time instant t_2 to time instant t_3 , a new norm-based dependency is set due to the insertion of a new normative goal or the associated normative power.

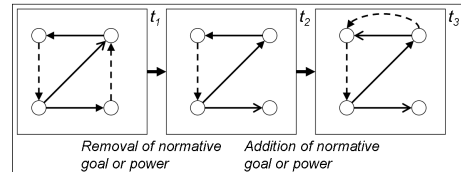


Figure 4. Norms' change.

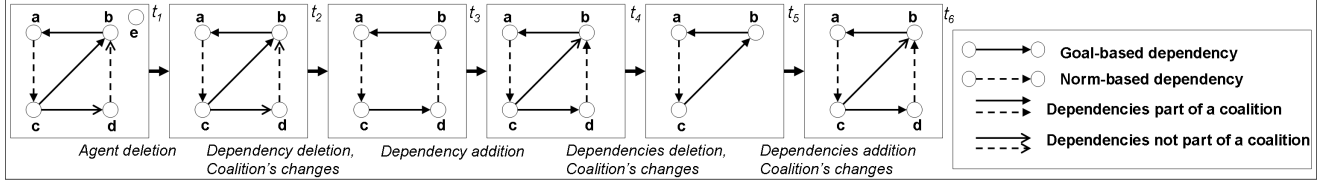


Figure 3. Agents and dependencies' change.

4.3. Coalitions' changes

The third kind of change is related to changes inside the coalition itself, e.g., an agent is excluded from a coalition because of a malicious behaviour. This third kind of change is the only one related to the coalition itself and it has to represent and measure the changes in the composition of each coalition of the system. We define a measure which calculates the ratio between the number of the goal-based and norm-based dependencies composing the coalition in each time instant and the dependencies composing the coalition in the previous time instant, as follows:

Definition 7 (Coalitions Measure): Let i be a time frame, N_i^{Coal} is given by the number of norm-based and goal-based dependencies of a coalition added to the network $(D_i^+ + O_i^+) \in C_i$ and deleted from the network $(D_i^- + O_i^-) \in C_i$ depending on the total number of norm-based and goal-based dependencies composing the coalition $(D_{i-1} + O_{i-1}) \in C_{i-1}$ at time frame $i - 1$:

$$N_i^{Coal} = \sum \left(\frac{(D_i^+ + O_i^+)c_i}{(D_{i-1} + O_{i-1})c_{i-1}} \right) + \sum \left(\frac{(D_i^- + O_i^-)c_i}{(D_{i-1} + O_{i-1})c_{i-1}} \right)$$

Example 3: Consider the coalition depicted in time instant t_1 of Figure 5. The coalition is composed by agents a , b and c . The passage from time instant t_1 to time instant t_2 sees the addition inside the coalition of agent d due to the reciprocity-based principle of coalition formation. From time instant t_2 to time instant t_3 , agent d is excluded from the coalition, without any change in the number or type of the dependencies composing the coalition itself. This can depend, as said, on a malicious behaviour of the excluded agent.

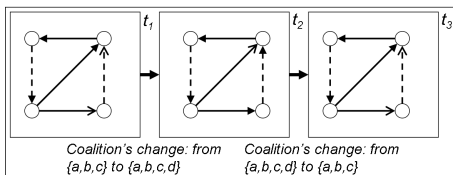


Figure 5. Coalitions' change.

The above measures are defined for one time moment only. We can unify these measures for a sequence of dependence networks associating to each time instant the

average number of changes. We can define this measure as follows:

Definition 8 (Changes Measures): Let i be a time frame of a sequence of social dependence networks, the measure of the changes' average is given by the fraction of the sum of the single measures and the number of available measures:

$$\frac{N_i^{Agent} + N_i^{Dep} + N_i^{Norm} + N_i^{Coal}}{\text{measures}}$$

Measures of example 1 vary as shown in Table 1.

	t_1	t_2	t_3	t_4	t_5	t_6
N_i^{Agent}	0/5	1/5	0/4	0/4	1/4	1/3
N_i^{Dep}	0/3	0/3	1/3	1/2	1/3	1/2
N_i^{Norm}	0/2	0/2	0/2	0/2	1/2	1/1
N_i^{Coal}	0/3	0/3	3/3	0/4	3/4	3/3
Changes	0	0,05	0,33	0,12	0,55	0,85

Table 1. Measures of Figure 3

Thanks to the changes measure, we underline that the two time frames with the main changes in comparison with their previous time frame are t_3 and t_5 , as can be supposed observing the relative figure. It can be noted that in our measures the deletion of a component increases the difference of the changes measure associated to two time frames in a row while the addition of these components causes a minor change. This behaviour is due to the relation of our measure with the game theoretical approaches for defining stability: the stability is maintained in order to avoid the breaking off of the agents from the grand coalition and form their own group.

We choose the simplest possible measures that capture the stability of the networks, because they represent all possible changes can be performed in the composition of coalitions and of the networks. When the average of the measures for a sequence of dependence networks presents a great difference in the values of two connected time instants, it underlines a lack of stability while when the average presents a small or inexistent difference between two connected time instants, the stability of the coalition and of the network in general is maintained. Moreover, the measures now only give a global indication of the stability of agents, dependencies, norms and coalitions. We could also measure whether changes in agents and dependencies coincides with changes in the coalition thanks to our four measures.

5. Related Work

In a multiagent perspective, a coalition can be viewed under two different representational frameworks. The first one regards cooperative game theory. Cooperative game theory studies those games in which players are able to make binding agreements with the aim to achieve a collective benefit. This approach is strictly related to the field of economics and various approaches of this kind have been presented in literature as, for example, the work of Shehory and Kraus [6]. The second perspective is based on the theory of the social power and dependence pioneered by Castelfranchi [4] as starting point and then developed in the context of coalition formation by Sichman [8] and Sauro [5]. This involves the development of a social reasoning mechanism that analyzes the possibility to profit from mutual-dependencies, e.g., two agents depend on each other for the satisfaction of a shared goal, or reciprocal-dependencies, e.g., two agents depend on each other for the satisfaction of two different goals. Both these two approaches present the following problems: they do not provide a modeling technique to represent coalitions' dynamics and to distinguish them.

6. Conclusions

We present a model to represent, at each time instant, the state of the system in terms of agents, goals, norms and the dependencies relating all these concepts. This model allows the distinction and measure of the possible coalitions' dynamics. In particular, we distinguish among three different kinds of coalitions' changes: changes based on addition or deletion of agents or goal-based dependencies, changes based on the addition or deletion of norm-based dependencies and changes on the internal structure of the coalition itself. It can be observed that with a more detailed model we could make more detailed and precise distinctions between the four kinds of changes. However, often we only have the given information, for example in systems' design, and we already would like to do this kind of analysis on these models. This is precisely where graph-theoretical social network techniques are useful. We combine these techniques with the normative multiagent paradigm introducing in the networks norm-based dependencies. The strength of this combination consists in building a modeling technique able to represent in an intuitive way not only the inter-relationships among the actors of the system but also external constraints such as norms and, particularly, obligations, e.g., in our Grid scenario. The main difficulty of this approach consists in the creation of a common model without simplifying too much the two original frameworks.

Moreover, we introduce four measures aiming to measure these changes inside the networks to each time instant and an average measure to compute the stability of a sequence of dependence networks. Our model allows to measure

coalitions' dynamics in terms of changing dependencies, agents and coalitions, distinguishing also among goal-based dependencies and norm-based ones. Using dependence networks as methodology to model a system advantages us from different points of view. First, they are abstract, thus they can be used for conceptual modeling, simulation, design and formal analysis. Second, they are used in high level design languages, like TROPOS [2], thus they can be used also in software implementation.

Concerning future work, we are working on a definition of coalitions' stability in our model, based on the presented measures, because of a lack of a definition of this notion in the field of social network theory. The notion of stability in our model can be identified intuitively in the absence of coalitions' changes we described but it is necessary to provide a formal definition of this notion and to associate it a measure able to represent it. Moreover, we start to simulate the use of our model and its associated measures in order to provide quantitative results based on our approach, similarly to social network theory approaches.

References

- [1] T. Ågotnes, W. van der Hoek, and M. Wooldridge. Temporal qualitative coalitional games. In *AAMAS*, pages 177–184, 2006.
- [2] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems Journal*, 8:203–236, 2004.
- [3] K. M. Carley. Dynamic network analysis. In *Dynamic Social Network Modeling and Analysis: Workshop Summary and Papers*, pages 133–145, 2003.
- [4] C. Castelfranchi. The micro-macro constitution of power. *Protosociology*, 18:208–269, 2003.
- [5] L. Sauro. *Formalizing admissibility criteria in coalition formation among goal directed agents*. PhD thesis, University of Turin, 2005.
- [6] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101:165–200, 1998.
- [7] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2008.
- [8] J. S. Sichman. Depint: Dependence-based coalition formation in an open multi-agent scenario. *Artificial Societies and Social Simulation*, 1(2), 1998.
- [9] J. S. Sichman and R. Conte. Multi-agent dependence by dependence graphs. In *AAMAS'02*, pages 483–490, 2002.
- [10] S. Villata. A normative multiagent approach to requirements engineering. *The logic journal of the IGPL*, 2009.

Dealing with incomplete normative states

Juan Manuel Serrano
 University Rey Juan Carlos
 Madrid, Spain
 juanmanuel.serrano@urjc.es

Sergio Saugar
 University Rey Juan Carlos
 Madrid, Spain
 sergio.saugar@urjc.es

Abstract—This paper puts forward a normative framework for computational societies which enables the handling of incomplete knowledge about normative relations. In particular, attempts to perform a social action are evaluated as permitted, prohibited (i.e. not permitted) or *pending* for execution (i.e. neither permitted nor prohibited). This latter category of attempts can eventually be resolved as permitted or prohibited attempts using the speech acts *allow* and *forbid*. We make use of the support for incompleteness of action language K in the formalisation of the framework. The proposal will be illustrated with some scenarios drawn from the management of university courses.

I. INTRODUCTION

Empowerments and permissions are two common normative devices in the design of computational societies [1], [2], [3], [4], [5]. The former notion allows us to model the institutional capabilities ascribed to agents of the society; the latter one serves to represent those desirable institutional states or courses of action which do not lend themselves to violation. The relations between both notions are commonly considered application-dependent, i.e. in some domains permission may be a necessary condition for empowered agents to act, whereas in others empowerment alone may be a sufficient condition. Concerning permissions, a difference is also made between regimentation and enforcement mechanisms in the implementation of normative systems [6]: regimented infrastructures (e.g. AMELI [7]) effectively prevent agents from executing some action if the corresponding permission does not hold; on the contrary, systems based upon enforcement rely on a subsidiary normative corpus of checking and sanctioning rules to bias the behaviour of agents towards the desired courses of actions. Finally, a common assumption in the literature is to consider that both empowerments and permissions are necessarily either true or false.

This paper challenges this last assumption for the case of permissions. In particular, it considers those situations in which the designers of the computational society do not have enough knowledge so as to generate a complete set of permission rules for certain classes of actions. For instance, let us consider a computational society designed to support the management of university courses. As part of the resulting specification, empowerment and permission rules are defined which partially regulate the social processes of the application domain, namely assignments, examinations, tutoring, lecturing, and so forth. In particular, the following norms concerning the creation of assignment groups will be considered. Firstly, empowerment to *set up* a working group for some assignment is granted to any student of the course who has not yet passed

that assignment. The attempt of setting up an assignment group will be permitted if, and only if, the assignment has been published, the specified submission deadline has not yet passed and the student is not participating in any other working group for that assignment. Alternatively, instead of setting up her own working group, a student may *join* a working group set up by another colleague in order to collaborate with him. Empowerment conditions for joining assignment groups coincide with those identified for the setting up of new groups. Some of the permission conditions for setting up working groups are also relevant, although only as necessary conditions. In particular, if some student attempts to join some working group and the corresponding submission deadline passed or she is already participating in another working group, then the attempt will be prohibited (i.e. not permitted). If none of these conditions hold, then there are no grounds for prohibiting the attempt. However, this does not mean that there are grounds for permitting the execution of the social action, since the ultimate decision on the permission or prohibition of the attempt lies with the initiator of the group (i.e. the student who set up the group). Thus, sufficient conditions for permitting or prohibiting the joining action can not be specified in advance by the designer of the society. In these scenarios, it would be very convenient to give the initiator of the group the possibility of either *allowing* or *forbidding* the corresponding social action so that the attempt is eventually permitted or prohibited.

This paper puts forward a formal model of permission which enables the representation of incomplete information about the normative status of social action attempts, such as the one commented above. Moreover, it formalises the meaning of the speech acts *allow* and *forbid* in the context of the previous model. In order to proceed with this formalization, we build on the notions of empowerment and permissions reported in [5]. That work introduces an operational semantics of computational societies using the action language C+ [8]. In this paper, we opt for the alternative action language K [9] due to its support for incompleteness.

The rest of the paper is structured as follows. Firstly, the most salient features of action language K for the purpose of this paper will be reviewed. Then, the general framework for social action processing will be introduced, describing the major features of the action description in language K. Next, the speech acts of allowing and forbidding will be formalised. Last, the major differences with previous work will be discussed and current work briefly described.

II. REVIEW OF ACTION LANGUAGE K

Action languages are formal techniques for representing and reasoning about the performance of actions in dynamic domains. The semantics of action languages is given in terms of transition systems, namely graphs whose states and arcs represent, respectively, the possible configurations of the domain and its evolution due to the concurrent execution of a set of actions. Commonly, action languages such as C+ describe transitions between states of the *physical* world, i.e. states which represent complete configurations of the domain, where each fluent is necessarily either true or false. In contrast, action language K [9][10] allows us to describe transitions between states of *knowledge*, where the truth values of some fluents may be neither true nor false, but unknown. The motivation behind action language K was thus to support agents with an incomplete view of the world in their planning processes.

An action description in language K is composed of a set of fluent and action declarations, a set of causation rules and a set of executability conditions:

- A fluent or action p is declared using an expression of the form:

$$p(X_1, \dots, X_n) \text{ requires } t_1, \dots, t_n$$

where X_i are variables and t_i are positive literals (i.e., true atoms) which specify the types of the corresponding variables¹.

- Causation rules are expressions of the form:

$$\text{caused } f \text{ if } B \text{ after } A$$

If the subexpression f is a fluent literal, the causation rule expresses that f is known to be true in the current state if B holds in the current state and A also holds in the preceding state. The subexpressions B and A are actually sequences of literals, possibly prefixed with the default (or weak) negation operator `not`. The expression `not f` holds if f is not known to be true, whereas the expression `not -f` holds if f is not known to be false. If both expressions hold then the truth value of f is *unknown*.

- The subexpression B can only refer to type or fluent predicates, whereas A can also refer to action predicates. If sequences B and A are empty, the corresponding `if` and `after` parts can be dropped from the expression. If the `after` part is empty the rule is called *static*, otherwise the causation rule is *dynamic*. Moreover, if f is the atom `false`, the causation rule represents an static (resp. dynamic) integrity constraint which allows us to filter out from the transition system ill-formed states (resp. transitions). Dynamic rules can be used to represent the non-executability conditions and effects of actions. In particular, the following macro rule is a shorthand of a dynamic constraint to represent that condition B blocks the execution of action a [10, sec. 2]:

$$\text{nonexecutable } a \text{ if } B$$

¹This is actually a slightly simplified version of this construction. See [10] for the full version and the meaning of *type* predicates.

- Executability statements are primitive (i.e. not macro) expressions of the form

$$\text{executable } a \text{ if } B$$

This kind of declaration expresses that action a is *eligible* for execution in any state of knowledge in which B holds. If we want the execution of action a to be not only possible but also mandatory, then a dynamic constraint can be declared. Since this a common requirement, this paper introduces the following macro rule which allows us to declare B as a sufficient condition for executing a :

$$\text{executed } a \text{ if } B \Leftrightarrow \begin{array}{l} \text{executable } a \text{ if } B \\ \text{caused false after not } a, B \end{array}$$

III. SOCIAL ACTION PROCESSING

Departing from its original motivation, action language K will be used in this paper for describing transitions between states of *institutional* worlds, rather than states of *knowledge* of some planning agent. In particular, the technical apparatus of language K will be exploited to represent institutional states where some normative fluents (e.g., permissions) may have an inherent, non-epistemic indeterminacy. In software engineering terms, the dynamic domain to be modeled is thus the social middleware infrastructure in charge of the management of the institutional state of the computational society, rather than the software components participating as agents in the society. The corresponding action description is partitioned in several sub-specifications corresponding to the different types of social entities of the computational society, namely *social interactions*, *agents* and *social actions*. A complete account of this specification, however, is beyond the scope of this paper². Instead, the focus here will be on the major features concerning the processing of social actions.

A. Social interactions

The institutional state of computational societies is hierarchically structured in terms of a tree of *social interactions*. The root of this tree, or top-level interaction, represents the social context within which the whole agent activity takes place; the other sub-interactions represent the social contexts for particular joint activities (i.e. social processes). For instance, assignment groups are represented by social interactions which are sub-interactions of courses, another type of social interaction. Social interactions may be *initiated* within the context of some other interaction, and eventually *finished* by the social middleware. The conditions which cause the execution of these actions are, in general, dependent on the type of interaction. Thus, university courses are automatically initiated when the new academic year begins, and assignment groups are initiated when some student successfully declares its initiation through the performance of the *set up* social action – in accordance with the empowerment and permission rules of the society. In this latter case, the *initiator* of the social interaction can be defined as the performer of the *set up* action.

²But see the C+ version [5] of the specification (which is not able to handle incomplete states), and the full K implementation available from <http://zenon.etsii.urjc.es/~jserrano/speech/k-impl.tgz>

```

fluents:
  state_i(I,S) requires
    interaction(I), interaction_state(S).
  context_i(I1,I2) requires
    interaction(I1), interaction(I2).
  initiator(I,A) requires
    interaction(I), agent(A).
  ...
actions:
  initiate(I1,I2) requires
    interaction(I1), interaction(I2).
  ...
always:
  nonexecutable initiate(I1,I2) if
    state_i(I1,open).
  caused state_i(I1,open) after
    initiate(I1,I2).
  caused context_i(I1,I2) after
    initiate(I1,I2).
  ...

```

Listing 1. K-specification of social interactions

Listing 1 shows some relevant features of the specification in language K of social interactions. In particular, it shows the declaration of fluents `state_i`, `context_i` and `initiator`, which represent, respectively, the run-time state of social interactions (either open or closed, values of the `interaction_state` predicate defined elsewhere), its interaction context and the possible initiator. Also shown is the declaration of the `initiate` action together with its non-executability conditions and effects (lines 14–19). These rules are declared within the scope of the `always` section, since they apply to every possible institutional state.

B. Agents

Agents are software components which interact through the social middleware as members of a given social interaction context, with the purpose of achieving some goal. In order to do so, they are empowered to perform social actions such as *setting up* new interactions, *joining* existing interactions, and so forth. For instance, the purpose of students is to pass the course to which they belong as members. In order to achieve this goal, students are empowered to set up working groups or to join existing ones in order to carry out some mandatory assignment. In case that the purpose of agents is too complex, its whole activity may be arranged in terms of a role-playing hierarchy of further agents. Thus, the activity of students within the context of working groups is represented by a new kind of agent role played by the course student. Agent roles are *played* and *abandoned* by the social middleware according to certain conditions. For instance, a course student role is automatically abandoned as soon as the agent passes the subject; a working group student is automatically created for the initiator of the working group, and for any student who successfully manage to *join* a pre-existing working group.

Listing 2 partially shows the K-specification of the agent

```

fluents:
  state_a(A,S) requires
    agent(A), agent_state(S).
  context_a(A,I) requires
    agent(A), interaction(I).
  player(A1,A2) requires
    agent(A1), agent(A2)
  ...
actions:
  play(A,I) requires
    interaction(I), agent(A).
  play_for(A1,A2,I) requires
    agent(A1), agent(A2), interaction(I).
  ...
always:
  nonexecutable play(A,I) if
    state_a(A,playing).
  caused state_a(A,playing) after
    play(A,I).
  caused context_a(A,I) after
    play(A,I).
  executed play(A1,I) if
    play_for(A1,A2,I).
  caused player(A1,A2) after
    play_for(A1,A2,I).
  ...

```

Listing 2. K-specification of agents

type, which includes the declaration of general fluents and actions shared by any kind of agent. Particularly, it shows the declaration of the fluents `state_a`, `context_a` and `player`, which represent the run-time state of agents (playing or abandoned), the social interaction context to which the agent belongs and its player agent, respectively. Also, it shows the declaration of the actions `play` and `play_for`. The former one causes some agent to be created within some social interaction context. The general specification only includes its non-executability condition and effect (lines 16–21). The action `play_for` causes some agent to be played (line 22) for a particular player agent (line 24).

C. Social actions

The activity of agent components within a multiagent society manifest itself through the performance of *attempts*. This external action allows an agent component to perform a given *social action*, namely to *say* something, *manipulate* the environment or *observe* the current state of some social entity. Due to lack of space, this paper will refer only to speech acts and, particularly, to declarations such as *set up* and *join*. Nevertheless, the processing of attempts by the social middleware is independent of the kind of social action. This process takes into account the *empowerments* and *permission* rules of the society. In particular, empowerments shall represent the institutional capabilities of agents, i.e. which social actions a given agent is capable of performing; permissions shall denote the circumstances under which these institutional capabilities can be exercised. Attempts by agent components are processed according to the following procedure:

- If the agent is empowered to perform the specified social action, then the attempt will be taken into account; otherwise, i.e. either if it is known for certain that the agent is not empowered, or it can not be concluded that it is empowered, the external action will be dismissed. In this latter case, the institutional state of the multiagent society will not be altered at all.
- If the agent is empowered to perform the action, but it is known that the specified performer is not permitted to perform it (i.e. it is prohibited), then the process is finished with a *prohibited* attempt status. On the contrary, if the agent is both empowered and permitted, then the social action is performed by the middleware. The effects caused through this execution depend on the kind of social action being performed.
- If the agent is empowered to perform that action, but it is neither known that the action is permitted nor prohibited, then the social action is kept in a *pending* state. This state will be eventually resolved into a *performed* or *prohibited* state as soon as it is known whether the action is permitted or prohibited.

This procedure is formalised as part of the *social action* type specification, whose major features are shown in listing 3. The signature of this specification includes the action declaration `attempt(Act,A)`, which represents the attempt made by some agent component *A* to perform the social action *Act*. This action is exogenous, i.e. its cause is to be found outside the system being modeled; correspondingly, it is unconditionally declared as `executable` (cf. line 20).

The different scenarios described above concerning the processing of attempts are modeled through different groups of static and dynamic rules. Firstly, if the intended agent is empowered to perform the social action then it will be declared as its performer (line 22), irrespective of the permission status. Empowerments, permissions and the performers of social actions, are represented by the fluents `empowered`, `permitted` and `performer`, respectively. If the agent attempting to perform the social action is empowered then the social action will be brought about in the next state to one of three execution states (represented by the fluent `state_sa`): `pending`, `prohibited` or `performed` (the possible values of the type predicate `social_action_state`).

- Firstly, if it is known that the action is not permitted (i.e. `-permitted(Act,A)`) then the resulting state will be `prohibited` (line 24).
- Secondly, if it is permitted then the action will be `performed` (line 27). Execution of actions is represented by the action `perform`, whose only effect at this level of abstraction is the change in the run-time execution state (line 39).
- Last, if the social action is neither known to be permitted nor prohibited, then the social action is kept in a `pending` execution state in the resulting system state (line 30). Eventually, the circumstances may change in such a way that the social action is known to be permitted

```

fluents:
  state_sa(Act,S) requires
    social_action(Act),
    social_action_state(S).
  context_sa(Act,I) requires
    social_action(Act), interaction(I).
  performer(Act,A) requires
    social_action(Act), agent(A).
  empowered(Act,A) requires
    social_action(Act), agent(A).
  permitted(Act)
    requires social_action(Act).
  ...
actions:
  attempt(Act,A) requires
    social_action(Act), agent(A).
  perform(Act) requires
    social_action(Act).
always:
  executable attempt(Act,A).
  ...
  caused performer(Act,A) after
    attempt(Act,A), empowered(Act,A).
  caused state_sa(Act,prohibited) after
    attempt(Act,A), empowered(Act,A),
    -permitted(Act).
  executed perform(Act) if
    attempt(Act,A), empowered(Act,A),
    permitted(Act).
  caused state_sa(Act,pending) after
    attempt(Act,A), empowered(Act,A),
    not permitted(Act), not -permitted(Act).
  ...
  caused state_sa(Act,prohibited) after
    state_sa(Act,pending), -permitted(Act).
  executed perform(Act) if
    state_sa(Act,pending), permitted(Act).
  ...
  caused state_sa(Act,performed) after
    perform(Act).

```

Listing 3. K-specification of social actions

or prohibited. In those cases, the social action will be resolved to the execution or the prohibition states by the corresponding rules (lines 34–37). Otherwise, the social action will persist until the performer agent is abandoned (i.e. its run-time state is changed to abandoned) or the interaction context is closed.

IV. FORBIDDING AND ALLOWING SOCIAL ACTIONS

Those social actions pending for execution will be resolved as prohibited or permitted attempts as soon as the rules of the society enables a definite conclusion on its permission status. As a complementary mechanism, particularly useful in the absence of general rules, run-time agents may also change the permission status through the speech acts *allow* and *forbid*³.

³Of course, since *allow* and *forbid* are speech acts, their performance is also governed by the corresponding empowerment and permission rules. For instance, initiators of assignment groups are unconditionally empowered and permitted to allow other students to join their groups. In other application domains, however, it may happen, for instance, that some agent is required to allow other agent to allow some other agent to do something.

```

fluents:
  new_role(Join ,A) requires
    join(Join), agent(A).
  ...
always:
  executed play_for(A1,A2,I) if
    join(Join), perform(Join),
    context_sa(Join ,I),
    performer(Join ,A2), new_role(Join ,A1).
  ...

```

Listing 4. K-specification of the *join* social action

```

fluents:
  action(Allow ,Act) requires
    allow(Allow), social_action(Act).
  ...
always:
  nonexecutable attempt(Allow ,A) if
    allow(Allow), action(Allow ,Act),
    not state_sa(Act ,pending).
  caused permitted(Act) after
    allow(Allow), perform(Allow),
    action(Allow ,Act).
  ...

```

Listing 5. K-specification of the *allow* social action

This section provides a formal account of the meaning of these speech acts and illustrates the formalisation of the assignment group scenario with the execution of a planning query. To account for a complete example, besides the *allow* and *forbid* speech acts, the *join* declaration will also be formalised.

A. Formalizing social actions

The specification of a new type of social action t_1 proceeds, firstly, by declaring a rule `social_action(x) :- t_1(x)`. This rule establishes that any entity of the new type shall be regarded as a social action, so that the rules which define the general structure and dynamics of social actions (cf. listing 3) are applicable for entities of that type. Secondly, new fluents representing the additional arguments of the new social action type must be declared. Last, new rules for representing the post-conditions of the performance of the new type of action, as well as their additional non-executability conditions, etc., have to be declared as well.

For instance, listing 4 shows the formalisation of the *join* declaration. By performing this speech act, the speaker declares that a new role is played within some interaction by it. The interaction and the speaker are represented by generic fluents of social actions, namely `context_sa` and `performer`. The new role to be played is declared as a new fluent, `new_role`, pertaining to this kind of speech act. The rest of the specification includes the particular effects associated to the execution of this kind of declaration, which are indirectly achieved through the internal action `play_for` (cf. listing 3).

Figure 5 shows the partial specification of the *allow* speech act. In this case, the generic social action specification is extended with the new fluent `action_a`, which represents the social action targeted by the *allow* speech act. The specification includes a condition which establishes that the social action to be allowed must be pending for execution (line 6). The effect of performing the *allow* action is to explicitly cause that the social action is permitted (9). The specification of the *forbid* speech act is similar to the one shown in listing 5. The only major difference pertains to its post-condition, which in this case resorts to the strong negation operator, i.e.

```

caused -permitted(Act) after
  forbid(Forbid), perform(Forbid),
  action_f(Forbid ,Act)

```

```

initially:
  -has_state_sa(join1). -has_state_a(s21).
  -has_state_sa(allow1).
always:
  state_a(s1 ,playing). context_a(s1 ,top).
  state_a(s2 ,playing). context_a(s2 ,top).
  state_i(wg1 ,open). initiator(wg1 ,s1).
  state_a(s11 ,playing). context_a(s11 ,wg1).
  player(s11 ,s1). empowered(s2 ,join1).
  action_a(allow1 ,join1).
  empowered(allow1 ,s11). permitted(allow1).
goal:
  member(wg1 ,A), player(A ,s2)? (3)

```

Listing 6. Working group scenario

B. Planning query

This section illustrates the semantics of the previous speech acts through a simplified implementation of the working group scenario. This implementation, shown in listing 6, features a consistent situation where the top-level interaction represents the university course to which two student agents, s_1 and s_2 , belong as members. The university course has a single working group wg_1 , previously set up by student s_1 . The activity of this student within the working group is represented by the role s_{11} . The query posed to the DLV^k planner (an implementation of action language K as a front end to the DLV answer set programming framework [9], [11]) asks for the possible ways in which the student s_2 may play a role within the working group wg_1 , in exactly three planning steps.

The output of the DLV^k planner is shown below. As expected, the first action that needs to be performed is an attempt by agent s_2 to join the working group. Two additional objects have to be declared in the scenario in order for this action to be performed: a *join* social action, $join_1$, and the agent to be played within the working group, s_{21} . These objects initially belong to the pool of objects which are available for the planning process⁴. Since the student is empowered to perform the join action but no permission rules are declared, the attempt to perform it results in a *pending* status. The next state features an attempt by agent s_{11} to allow the performance of the join action, namely to perform action $allow_1$. Since this

⁴Formally, these are objects which have no state, e.g. agents which are being neither played nor have been abandoned.

agent is both empowered and permitted to perform that social action, the permissions to execute the $join_1$ action are in effect in the next state. This, in turn, causes the performance of the join action and the consequent playing of the student agent within the working group.

```
STATE 0: state_a(s2,playing) state_i(wg1,open)
         empowered(s2,join1) new_role(join1,s21) ...
ACTIONS: attempt(join1,s2)
STATE 1: state_sa(join1,pending) performer(join1,s2)
         empowered(s11,allow1) permitted(allow1)
         action_a(allow1,join1) ...
ACTIONS: attempt(allow1,s11) perform(allow1)
STATE 2: state_sa(allow1,performed)
         state_sa(join1,pending) performer(join1,a1)
         permitted(join1) ...
ACTIONS: perform(join1) play_for(s21,s2,wg1) play(s21,wg1)
STATE 3: state_a(s21,playing) player(s21,s2)
         context_a(s21,wg1) ...
```

V. DISCUSSION

The model of empowerment and permission put forward in this paper contrasts with other approaches based on ASP [3], [12], the event calculus [4] or action languages [2] in several respects. Firstly, the subjects of empowerments and/or permissions in these approaches are *events* which represent the observable or institutional actions to be evaluated. Moreover, normative fluents are boolean so that these events are evaluated in a single transition step either as permitted or prohibited. In contrast, permissions are applied in our framework to a particular kind of social *entity*, viz. social actions, which can be assigned a permitted and prohibited status, but also an unknown one. Thus, our framework does not force the designer of the computational society to add a complete set of permission rules.

Secondly, the strong negation operator allows the designer to explicitly declare prohibition rules, whereas other approaches have to resort to the default “everything which is not permitted, is prohibited”, which may not always be adequate. In particular, explicit prohibitions are very convenient in order to represent necessary conditions of permission rules. For instance, the following rule states that a necessary condition to join some working group is that the deadline for submitting the corresponding assignment has not passed yet:

```
caused ¬permitted(Join) if
  context_sa(Join,W), working_group(W),
  assignment(W,A), deadline(W,D),
  current_time(T), D<T.
```

The assignment group scenario also served to illustrate a situation where empowerments and permission rules concerning a single type of social action are, respectively, complete and incomplete. This represents a good case in favour of the distinction between empowerments and permissions, which some approaches neglect (dispensing with one of the two notions).

Lastly, two normative social actions, *allow* and *forbid*, are smoothly introduced within the normative framework in order to handle those situations of incomplete normative knowledge. The semantics proposed for these actions is aimed at particular cases that can not be solved using the general normative

knowledge of the society. This ad-hoc character tallies well with the natural language meaning of the corresponding English speech act verbs [13].

The normative framework reported in this paper is part of a larger research project aimed at the specification of a language for programming social applications [5], viz. software systems designed to support human interaction in arbitrary social contexts. This broad class of target applications include common online communities, but also other software systems deployed in more specialised settings such as business process management. This general goal partly explains some of the features of the proposed normative framework, such as its bias towards regimentation. Current work focuses on extensions to support commitments, an essential construct for many social application domains.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their detailed comments. Research sponsored by the Spanish MICINN, project TIN2006-15455-C03-03.

REFERENCES

- [1] A. J. I. Jones and M. J. Sergot, “A formal characterisation of institutionalised power,” *Logic Journal of the IGPL*, vol. 4, no. 3, pp. 427–443, 1996.
- [2] A. Artikis, M. Sergot, and J. Pitt, “Specifying norm-governed computational societies,” *ACM Transactions on Computational Logic*, vol. 10, no. 1, 2009.
- [3] O. Cliffe, M. D. Vos, and J. A. Padget, “Answer set programming for representing and reasoning about virtual institutions,” in *CLIMA VII*, ser. Lecture Notes in Computer Science, K. Inoue, K. Satoh, and F. Toni, Eds., vol. 4371. Springer, 2006, pp. 60–79.
- [4] N. Fornara and M. Colombetti, “Specifying artificial institutions in the event calculus,” in *Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, V. Dignum, Ed. IGI Global, 2009, ch. 14, pp. 335–366.
- [5] J. M. Serrano and S. Saugar, “Run-time semantics of a language for programming social processes,” in *9th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA IX)*, ser. Lecture Notes in Artificial Intelligence, M. Fisher, F. Sadri, and M. Thielscher, Eds., vol. 5405. Springer, 2009, pp. 37–56.
- [6] D. Grossi, *Designing Invisible Handcuffs*. SIKS Dissertation Series No. 2007-16, 2007.
- [7] M. Esteva, B. Rosell, J. A. Rodríguez-Aguilar, and J. L. Arcos, “AMELI: An agent-based middleware for electronic institutions,” in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, vol. 1, 2004, pp. 236–243.
- [8] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner, “Non-monotonic causal theories,” *Artif. Intell.*, vol. 153, no. 1-2, pp. 49–104, 2004.
- [9] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres, “A logic programming approach to knowledge-state planning, ii: The dlv^k system,” *Artif. Intell.*, vol. 144, no. 1-2, pp. 157–211, 2003.
- [10] —, “A logic programming approach to knowledge-state planning: Semantics and complexity,” INFSYS Research Report, Tech. Rep. 1843-01-11, Oct. 2002.
- [11] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello, “The dlv system for knowledge representation and reasoning,” *ACM Trans. Comput. Log.*, vol. 7, no. 3, pp. 499–562, 2006.
- [12] M. Gelfond and J. Lobo, “Authorization and obligation policies in dynamic systems,” in *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings*, ser. Lecture Notes in Computer Science, M. G. de la Banda and E. Pontelli, Eds., vol. 5366. Springer, 2008, pp. 22–36.
- [13] A. Wierzbicka, *English speech act verbs. A semantic dictionary*. Australia: Academic Press, 1987.

Directing Status Messages to their Audience in Online Communities

Milan Stankovic, Philippe Laublet

LaLIC

Université Paris IV – Sorbonne, Paris, France

milan@milstan.net, philippe.laublet@paris-sorbonne.fr

Alexandre Passant

Digital Enterprise Research Institute

NUI-Galway, Galway, Ireland

alexandre.passant@deri.org

Abstract— In this paper we present the results of our user study about status message sharing on the Social Web. The study revealed the privacy and information noise (sometimes originating from gap of understanding and sometimes from lack of significance) to be the key problems in the domain and allowed us to unveil their nature. Further on we present the existing solutions and workarounds for those problems and introduce the idea that Semantic Web technologies could help confront those problems in a more complete way. We propose a way to use semantic descriptions of status messages, their intended audiences and distributed data about users to direct status messages to their intended recipients. Particularly, we rely on the Online Presence Ontology as a vocabulary for exposing status message semantics, and we provide necessary extensions to support status message directing.

Index Terms—Faceted Identity, Linked Data, Online Presence, Social Web, Social Networks.

I. INTRODUCTION

STATUS messages are short textual expressions that describe the state of a user's presence in the online world. Sharing status messages on different social services on the Web (Microblogging services, Instant Messaging platforms, Social Networks) became a common practice for people to share thoughts, feelings of the moment, announce one's presence in the online world and broadcast information. However, as more and more users take part in status message sharing, the open communities become overloaded with status updates. Many problems arise from such an overload. Firstly, confidentiality of status messages in open communities is a significant question, since not all status messages are meant for general public. Some should be kept private from certain contacts who might use them in an inappropriate way. An example could be a status message revealing somebody's drinking habits, meant to amuse personal friends, but the same status message could be a source of inconvenience if shown to work colleagues.

Apart from private nature of some status messages there are other reasons why a particular status message might not be

suitable for a certain audience. For example, some status message updates may have no significance for certain groups of contacts that consider them as information noise. It is a common case that we subscribe to someone's statuses because of the interest in professional news he/she is sharing, but aside we get a lot of postings about the person's personal life that don't interest us. Problems like those limit in a great deal, the usefulness of today's status sharing services (mostly microblogging services and Social Networks)

In order to explore more deeply the nature of the problem of confidentiality and other key problems in status message sharing in large communities on the Social Web we conducted a qualitative user study with subjects who are using status messages for different purposes and in different contexts. The goal of the study was to develop understanding of the key problems, factors that make a status message open or confidential – that determine its intended audience. Apart from understanding the problems, the study allowed us to explore the space of possible solutions. In Section 2 we present the results of our user study. Section 3 presents the Presence Diamond, a useful notion for the study of presence online as a faceted phenomenon. Section 4 lists currently available solutions for problems identified in the study. In Section 5 we introduce a way to direct a status message to its intended audience using Semantic Web technologies, and we show how those technologies are flexible to support even dynamic audience definitions (where members of the audience change frequently). Section 6 presents related work and in Section 7 we conclude the paper.

II. THE USER STUDY

The user study was conducted through a series of ten interviews with users of social networks and microblogging platforms who have been using them for status message sharing for some time (a year in average). The 30-35 minute interviews were field-noted and audio recorded for further reference. Users' age ranged from 22 to 35. This choice proved to correspond well to demographics of users of the most active microblogging services (documented in a statistical report done by Pew Internet¹). Equal number of male and female subjects, from France and Ireland, with different

Manuscript received July 30, 2009.

The work presented in this paper has been funded in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2).

¹<http://www.pewinternet.org/Reports/2009/Twitter-and-status-updating.aspx>

origins and backgrounds, took part in the interviews.

After a couple of questions about users' background, users were asked to tell their status message publishing experiences. The main goal was to identify their context in the time of publishing, nature of the status message content and the intended audience. The inconveniences and the inability of microblogging tools and social networks to meet their status message sharing needs were also explored.

Once we collected the user stories, we relied on Grounded Theory inspired approach to extract relevant categories from them, and further generalize the categories to super-categories that we call – major issues. Open and Axial coding were used with participation of two researchers in order to reduce the impact of subjectivity.

Grounded Theory was introduced by Glaser and Strauss [1] and has served ever since for analysis of results in qualitative research in Social Sciences. Grounded Theory is an approach to looking systematically at qualitative data to derive codes and group them into relevant categories that will further be generalized into concepts that make the ground for generating a theory. Generalizations are derived by thinking efforts of researchers. Due to a space limit, in this paper we present only a part of our findings - the highest level generalizations, and we briefly describe them with some of the lower level generalizations that we find the most relevant to our intended readers.

Generally we discovered that many times when users publish a status message, they have a certain audience in mind. The status message is intended for a particular audience either because of its ability to understand the message (or the inability of others to understand it properly) either because of significance of the message for a certain group (and insignificance for others) or because of the confidential nature of the status message content. The next three sections present those major issues – reasons why a status message has its particular audience.

A. Gap of Understanding

In many cases where a certain status message is not meant for a certain group of people it is because of their inability to understand, properly interpret and maybe even reply to the content of the message. Sometimes the inability arises from **shallow acquaintance** like in cases where the user publishing a status messages knows a certain group of people for a short time. The shallowness of acquaintance can be an obstacle for this group of people to understand jokes, metaphors and properly interpret the intended meanings of status messages. Sometimes the gap of understanding results from **lack of competence** like in cases where users use status messages to ask for advice, or provoke professional discussions. This problem is also present in scenarios of automatic postings of status messages across services (e.g. automatic forwarding from Twitter to Facebook) where mostly different audiences are present on different services. Quite often personal friends from one service (Facebook in our case) don't understand and find irrelevant the profession-related status messages posted on

another service (Twitter in our case).

Some status messages bear a **socially established meaning**, understood by a small community of people, like those containing internal jokes, or internal aliases and metaphors. Such status messages may be misinterpreted by people outside that small community and may be source of misunderstandings, inappropriate comments and other inconveniences.

B. Lack of Significance

In other cases, a status message is not intended for some people simply because they have no interest in it. This is the case when a status message relates to a certain domain and thus can be of significance only to people with an **interest in the domain**. This case is common when people make connections based on a shared interest, stay in touch and then use status messages to spread domain related news, announce events and provoke discussions. In some cases it is the interest in the domain that makes a certain group of people not interested in other non domain-related status messages of a user. For people who are not familiar with the domain such messages can represent noise.

In other cases some groups of people might not be able to **make use of the information** in the status message which has an informative purpose. This is the case with status messages highly dependent on location – like those containing invitations to local parties and announcements of local events. In both cases such status messages are irrelevant to people from other locations who could not make use of the announcement.

C. Privacy

Privacy is an issue that occurs when a user wants to explicitly restrict access to some groups of contacts for some types of status message or even only for a particular status message. It is usually related to groups of higher granularity, like the case of separating status messages for work and private contacts. People usually perceive some content types (like feelings and moods or travel experiences) to be suitable only for closer contacts or contacts of a more private nature, while those status messages should be kept private from some other (more professional) groups of contacts.

Some users, on the other hand express concern about the **possibilities to track** their status messages to the past and draw conclusions about their personality which would be out of their control. The concern is expressed about the uncontrolled data integration possibilities across services and attempts to integrate status messages with other content about the user and thus perform some spy-like behavior.

III. THE PRESENCE DIAMOND

Once we acknowledge that many status messages have an intended audience and that access to them should in some cases be restricted to that particular audience (in cases of confidential messages), it becomes clear that one user might have different status messages for different audiences at the

same time.

In fact, emitting different information (appearances) to different groups of observers is not restricted to status messages, but spans the whole notion of online presence. By the term online presence we refer to the totality of information that allows perceiving one's presence in online communities. Apart from status messages as an element of presence, availability for interaction might also have a faceted nature and be different for different groups at different times. One can easily imagine a working situation where a user is available for interaction only with his work colleagues and busy for all the others. Access to different presence information might also be given only to specific groups of contacts (like in the case of sharing the current location only with closest friends).

Therefore, there is a need to look at the notion of online presence as a faceted phenomenon. For this reason we introduce the notion of the presence diamond (Figure 1.) to capture the faceted nature of presence and the need to appear differently to different groups of people.

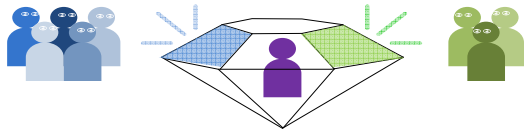


Figure 1. The Presence Diamond²

The notion of presence diamond allows us to look at a person's online presence as a diamond whereby different observers are introduced to different facets of the diamond. Facets differ among themselves in **different types of presence data** that is accessible by observers of a facet (like in cases where one group of observers can access a person's location, availability and a status message, and another group can access only the status message), **different granularity of data** (like in the case of sharing the exact location with closest friends and only the current city/country with strangers), and in **different data** that is emitted to different observers (like having different status messages and different availability for different groups of contacts).

Even though we focus on status messages in this paper, we will look at the problem of directing status messages to their intended audience as a sub-problem of enabling faceted online presence, and will therefore favor solutions general enough to address the faceted nature of presence as a whole.

IV. INCOMPLETE WAYS TO DEAL WITH STATUS MESSAGE DIRECTING

Some ways to direct status message updates to a particular audience already exist. In this section we present the workarounds found and applied by users, as well as solutions

developed as features of Social Web sites. For each of these solutions we discuss its incompleteness.

A. User Workarounds

Some users manage to separate their contacts on different Social Web services, by taking into account the nature of relationship with a particular contact. For example, a number of users maintain a list of work-related contacts on twitter while having a more personal network of friends on Facebook, and then share different status message updates for the different audiences. This way status messages related to private life can be kept confidential from work colleagues, and personal friends don't have to be bothered by work related postings. However, the fact that some contacts use only one social network stands in the way of such a separation. If some of the user's work colleagues use only Facebook, then maintaining the separation would mean not connecting at all with those persons. Apart from this limitation, if the separation by purpose is not done at the start, it is hard to impose it once the user has accepted different types of contacts to his/her social network.

Another way to deal with the identified issues is just to restrict oneself to publishing only status messages acceptable for the wide audience. Some users choose not to publish too personal status messages because work-related contacts might see them, and not to publish work-related status messages because they might not be of interest to their friends. This approach limits the potential of status message sharing in a great deal excluding many professional and staying-in-touch use cases.

B. Solutions developed by Social Web Sites

Solutions for niche microblogging and micro-broadcasting began to emerge recently. Those Social Web sites allow for broadcasting of status messages in closed communities (like in ShoutEm³) or to people gathered around a certain interest (like in Static⁴). However they mostly require intended recipients of the status message updates to join each closed community which can get quite complicated having in mind the number of intended audiences a user might have. This approach certainly leads to social network fatigue – a phenomenon of loss of motivation to participate in yet another social network when confronted with joining many social networks and building identities on them.

The new service E⁵ can be used to manage adding different people to different social networks according to the nature of the acquaintance (e.g. adding friends to Facebook and business contacts to MySpace). However, it is hard to enforce this separation since not all users are present on each of those networks and therefore some of connections might be lost if they do not meet the purpose one user has given to his/her social network account.

² The figure and the notion of the Presence Diamond are strongly inspired by the notion of the diamond of digital identity, that Mike Roch, Director of IT Services at University of Reading, introduced at the Eduserv Digital Identity Workshop in London, January 08, 2009

³ <http://www.shoutem.com/>

⁴ <http://www.static.com/>

⁵ <http://www.mynameise.com/>

V. THE LINKED DATA WAY

The term Linked Data [2] refers to publishing and interlinking structured data on the Web in RDF⁶ with the assumption that the value and usefulness of data increases the more it is interlinked with other data. This effort to publish the data online using open standards and interlink data sources is aimed at transforming the Web of documents towards a more (re)usable, machine readable Web of Data.

We argue that additional semantics describing a status message, as well as semantics (partially already published as Linked Data) describing users and their current context can be helpful to direct a status message to its intended audience, and thus reduce information noise and contribute to ensuring privacy. In particular we argue that currently available Linked Data sources can help define the intended audiences of status messages, relying on user properties described in those sources (interests, locations, social graph, etc.)

To enable publishing and exchange of such additional semantics, we decided to enrich an existing vocabulary - the Online Presence Ontology (OPO)⁷ - with the information about intended audience of a status message. The Online Presence Ontology presented in [3] provides a way to describe a user's current state of presence in the online world, including his/her availability for interaction, current status message, location and other elements of context. As such this vocabulary can be elegantly complemented with a way to direct a status message (or even the whole notion of Online Presence) to a certain audience. To enable this, we have extended the OPO with the notion of Sharing Space.

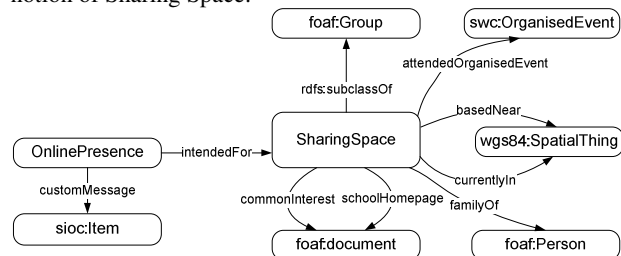


Figure 2. An excerpt from the Online Presence Ontology

A Sharing Space, in our specification, is a group of people (or agents) with whom particular information can be shared. As shown on Figure 2, the `OnlinePresence`, encompassing (among other properties) the current status message of a user, can be connected to its intended audience through a property `intendedFor` by linking it with the notion of `SharingSpace`. The status message itself is represented using the `Item` concept from the `SIOC`⁸ [4] ontology in order to enable replies to the status message and make use of this concept's suitable semantics. `SharingSpace` is also enriched with a list of properties to allow representing of the common attributes that bound members of the `SharingSpace` together (e.g., common interest, common current location). In order to express the

semantics of those attributes we relied on concepts from widely used vocabularies (FOAF⁹, SWC¹⁰, WGS84¹¹). For more details about the ontology design we refer the readers to the project website and the ontology specification¹².

By identifying people who are intended to receive a status message, the notion of `SharingSpace` can help software systems to deliver status messages to specific people (members of the `SharingSpace`) and thus deal with information noise and even ensure confidential status message exchange.

In order to properly define `SharingSpaces` according to the needs of real life scenarios, we will rely on the results of our user study, presented in Section 2. According to our study results, some of the major ways to define the intended audience are: friends of a certain friend; people having a certain interest; friends from a particular online community; people being in a certain location; people having a certain nature of relationship with the user; people who were affiliated in the same institution; and custom assembled groups of contacts.

A lot of information needed to define those groups (users' current and permanent locations, interests, friends' lists, etc.) is already available on the Social Web, and many sources already publish this data using vocabularies such as FOAF and SIOC [4]. Relying on those existing resources, `SharingSpaces` could be dynamically defined using simple SPARQL¹³ queries that could identify the members of a particular `SharingSpace` by collecting data across different data sources. We believe that this way of defining `SharingSpaces` is flexible enough to cover the needs of real life scenarios identified in our user study, and we will illustrate it on an example in the following subsection.

When proposing to use data from various distributed datasets, we should acknowledge that executing queries over distributed datasets might be a challenging task. However, this challenge has already attracted researchers to develop solutions for this distributed scenario. One of them is a system `DARQ` [5], an engine for federated SPARQL queries.

Apart from specifying `SharingSpace` members using SPARQL, the new version of the OWL language¹⁴, currently available as OWL 2 Working Draft [6] will provide a way to define `SharingSpaces` through richer restriction axioms such as property chains. Property chains would allow to state that if a user satisfies a certain property then he is automatically a member of a `SharingSpace`. We also believe that the emerging Rule Interchange Format [7] (currently a working draft) will be a useful way to define and exchange `SharingSpace` definition rules across different systems that may use different rule languages internally.

⁹ Friend-f-a-Friend vocabulary <http://xmlns.com/foaf/spec/>

¹⁰ Semantic Web Conference Ontology http://data.semanticweb.org/ns/swc/swc_2009-05-09.html

¹¹ World Geodetic System ontology <http://www.w3.org/2003/01/geo/>

¹² Other properties and classes introduced to support the notion of `SharingSpace` can be found in the specification document <http://www.milanstankovic.org/opo/specs/>

¹³ <http://www.w3.org/TR/rdf-sparql-query/>

¹⁴ <http://www.w3.org/2004/OWL/>

⁶ Resource Description Framework <http://www.w3.org/RDF/>

⁷ <http://www.milanstankovic.org/opo/>

⁸ <http://sioc-project.org/>

A. Scenario of Use

To better illustrate the flexibility of our approach and the usefulness of Linked Data, we present a scenario of publishing a status message together with a dedication to a particular Sharing Space. Figure 3 will serve as a graphical support to our explanations.

In this scenario, our example user Harry is organizing a reunion for his friends from the Semantic Web community. The reunion will take place in Paris, and Harry wants to announce it in his status message.

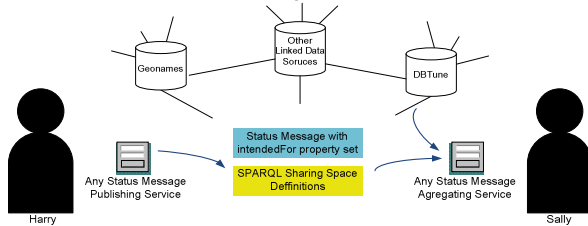


Figure 3. Publishing a Status Message

Thanks to the open nature of Semantic Web technologies, any status message publishing service (including microblogging platforms, social networks, chat platforms) can publish a status message and describe it using the OPO vocabulary. So, Harry's status message publishing service can make the semantically described message available to all status message consuming services. It can further associate it with a particular audience, by using the `intendedFor` property and the concept of a `SharingSpace`. Along with OPO data about the status message itself, Harry's service can publish a SPARQL query to define the members of the Sharing Space. In our case, since Harry's message is intended for people interested in Semantic Web who are currently in Paris, the SPARQL Query would look like shown in Figure 4.

To make better use of the data available in Linked Data sources, we can reuse existing URIs used by those sources. In our example we rely on the Geonames¹⁵ URI for Paris, to uniquely identify this geographical location.

Once the message is available together with its semantic description, and a Sharing Space definition, other services can consume it and make it available to their users. Let us take another example user, Sally. She is Harry's friend, interested in Semantic Web and currently visiting Paris (according to her last published status message with associated geographic location information). Although Sally is not using the same status message publishing service as Harry, her Social Network (SN) service, can retrieve semantically described status messages and SPARQL queries defining Sharing Spaces. Since information about Sally's interest is available in one of her FOAF files, and available to her SN, and since her current location is also known to SN, applying the SPARQL query from Figure 4 will put Sally in `SWPeopleInParis` Sharing Space - the one Harry's status message is intended for.

Sally's interface for browsing status messages can now

make sure that status messages intended for her get to her attention and somehow stand out from the abundance of other status messages put online by her friends and other people.

```
PREFIX opo: <http://http://ggg.milanstankovic.org/opo/ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
CONSTRUCT
{
  <http://example.org/ns#SWPeopleInParis>
    rdf:type opo:SharingSpace;
    foaf:member ?person.
}
WHERE
{
  ?person foaf:topic_interest
    <http://dbpedia.org/resource/Semantic_Web>.
  ?person opo:declaresOnlinePresence ?presence .
  ?presence opo:currentLocation
    <http://sws.geonames.org/2988507/>.
}
```

Figure 4. Example definition of a Sharing Space

B. Some Benefits of Sharing Spaces

Using the definitions of Sharing Spaces, like those shown in this paper, and publishing status messages that rely on the extended OPO vocabulary can help direct a status message to its audience. As opposed to solutions where particular (sometimes even closed) services are used to dedicate a status message to a certain group of people, our approach offers a way to dedicate a status message to a certain audience regardless of the service being used to publish them and present them. It is the use of widely accepted Semantic Web standards (e.g., RDF(S) and OWL) that make the intended audience specifications universal and thus applicable everywhere.

The approach also allows to take into account the ever changing nature of user-related data, since membership in a Sharing Space can be defined through a property and not by naming particular members. Therefore users can belong to a sharing space at one time when they satisfy a certain condition (e.g. currently located in Paris), and not belong to it at all other times.

Apart from combating status message overload and helping relevant messages to reach their audience, Sharing Spaces can serve as a ground for ensuring privacy and confidential status message sharing. Our approach is based on the idea [8] that ensuring trust and privacy on the future Web can be grounded on the interlinked graph of data (i.e. Linked Data) and policies that take advantage of existing data sources. The introduced change in the OPO vocabulary is a first step in this direction, allowing to specify the intended audience of a status message by reusing existing (linked) data on the Web. Further mechanisms to enforce the delivery of a status message to the specified intended audience can be built on top of our presented solution. The advantage of this approach is that dedicating a status message to its audience is quite a general solution, addressing at the same time the challenge of dealing with information noise, and being the ground for ensuring the confidential status message sharing.

¹⁵ <http://www.geonames.org/>

VI. RELATED WORK

Similar to our use of SPARQL to define sharing spaces i.e. intended audience groups, Alessandra Toninelli et al. [9] use RDF and SPARQL triple patterns to build social graph aware policies. Using triple patterns different policies can be created to grant access to user's attention (e.g., ring her phone). However this work is more related to mobile devices as it strongly reflects the specifics of communication using a mobile device, and in this sense it is complementary to our work in effort to make use of social data available in Linked Data sources to enhance user's interaction with devices and make her communications more adapted to her current situation. Another point of difference is that the socially-aware policy model is more concerned at granting/restricting access to a certain resource than dedicating/directing presence information to a certain audience.

VII. CONCLUSIONS AND FUTURE WORK

In this paper we presented the results of our user study, based on qualitative research techniques, which was aimed at identifying the nature of problems surrounding status message publishing. Our study emphasized the need to direct a status message to a particular audience in order to deal with major issues like: Lack of Understanding, Significance, and Privacy.

We have shown how users try to deal with those issues and what solutions did the Social Web sites come up with to help with directing a status message to a certain audience. However, we judged all those solutions as incomplete either because they require users to join particular status sharing networks or because they restrain users from publishing certain types of status messages.

Our solution to the problem of dedicating a status message to a particular audience is based on providing semantic descriptions of intended audience and taking advantage of existing data about users published as Linked Data on the Web. Particularly we rely on a small extension of the Online Presence Ontology that allows for associating the intended audience information to a status message. Since the solution is based on Semantic Web technologies it allows a high level of interoperability and gives the intended audience information the ability to flow across different status message sharing services. Moreover, our semantic descriptions of intended audiences possess the ability to collect the intended audience members information from different Linked Data sources across the Web, which makes them universal.

Our future work will consist in evaluating the practical aspects of our proposal by extending the distributed microblogging platform SMOB¹⁶, described in [10] to publish and take into account the intended audience information through the use of new notion of Sharing Space introduced in the Online Presence Ontology. The new version of SMOB will make use of data available as Linked Data on the Web to create refined descriptions of audience for its status messages.

While the simple publishing and taking into account of intended audience information would be sufficient to combat the information noise problem, encompassing both issues of Lack of Understanding and Significance; some additional access control mechanisms must be employed to ensure that the intended audience specifications are properly applied across the Web. As a solution to access control we are considering to use the FOAF + SSL protocol [11] – a lightweight solution for authentication and authorization, based on the semantics exposed using the widespread FOAF vocabulary. The OpenID¹⁷ framework for providing a single digital identity across the internet can also elegantly contribute to achieve simple access control. OAuth¹⁸ authorization protocol could also be helpful in ensuring secure exchange of intended audience information across different services on the Social Web.

Although our solution for directing a status message to its audience is flexible in specifying the intended recipients of the status message, a lot of work remains to be done to ensure that the unintended recipients do not get access to it. We see the presented extension of OPO and the notion of Sharing Space as a first step in this direction.

VIII. REFERENCES

- [1] Glaser, B. & Strauss, A. *Discovery of Grounded Theory*. Chicago: Aldine, 1976.
- [2] Berners-Lee, T. *Design Issues: Linked Data.*, 2006, <http://www.w3.org/DesignIssues/LinkedData.html>
- [3] Stankovic, M. *Modeling Online Presence*. In: *Proceedings of the First Social Data on the Web Workshop*, Karlsruhe, Germany, 2008.
- [4] Breslin, J.G., Harth, A., Bojars, U. & Decker, S. "*Towards Semantically-Interlinked Online Communities*", *Proceedings of the 2nd European Semantic Web Conference (ESWC '05)*, LNCS vol. 3532, pp. 500-514, Heraklion, Greece, 2005.
- [5] Quilitz, B., Leser, U.: *Querying Distributed RDF Data Sources with SPARQL*. In: *ESWC. (2008)* Available: <http://www.eswc2008.org/final-pdfs-for-web-site/qpII-2.pdf>
- [6] Motik, B., Patel-Schneider, P.F. & Parsia, B. *OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax*, 2008, <http://www.w3.org/TR/2008/WD-owl2-syntax-20081008/>
- [7] Boley, H., Hallmark, G., Kifer, M., Pasche, A., Pollares, A. & Reynolds, D. *RIF Core*, 2008, <http://www.w3.org/TR/rif-core/>
- [8] Passant, A., Kärger, P., Hausenblas, M., Olmedilla, D., Pollares, A. & Decker, S. *Enabling Trust and Privacy on the Social Web*, In *Proceedings of W3C Workshop on the Future of Social Networks*, 15-16 January 2009, Barcelona,
- [9] Toninelli, A., Khushraj, D., Lassila O., and Montanari, R.: *Towards Socially Aware Mobile Phones*. In: *Proceedings of the First Social Data on the Web Workshop*, Karlsruhe, Germany, October 27, 2008, *CEUR Workshop Proceedings*, ISSN 1613-0073,
- [10] Passant, A., Hastrup, T., Bojars, U. & Breslin, J. *Microblogging: A Semantic Web and Distributed Approach*. In: *Proceedings of the 4th Workshop on Scripting for the Semantic Web*, Tenerife, Spain, June 02, 2008, *CEUR Workshop Proceedings*, ISSN 1613-0073,
- [11] Story, H. *FOAF & SSL: Creating a global decentralized authentication protocol*, 2009. In *W3C Workshop on the Future of Social Networking*, 15-16 January 2009, Barcelona, Spain.

¹⁶<http://smob.sioc-project.org/>

¹⁷ <http://openid.net/>

¹⁸ <http://oauth.net/>

Managing water demand as a regulated open MAS. (Work in progress)

Vicente Botti¹, Antonio Garrido¹, Adriana Giret¹ and Pablo Noriega²

¹ DSIC, Department of Information Systems and Computation, Universitat Politècnica de Valencia,

² IIIA, Artificial Intelligence Research Institute, CSIC, Spanish Scientific Research Council,
{vbotti,agarridot,agiret}@dsic.upv.es, pablo@iia.csic.es

I. WATER MANAGEMENT AS A MAS

The management of natural resources is a challenge of significant social relevance. At the core of water policy is the need to foster a more rational use of the resource and this may be addressed by creating an efficient market of water rights [4]. However the design and operation of such a market is not an easy endeavor because it needs to coexist in a complex social and legal framework.

Most water management models are based on equational descriptions of aggregate supply and demand in a water basin [2] but few include an agent-based perspective. We explore an alternative approach in which individual and collective agents are an essential component because their behavior (and effects) may be influenced by policy-making. Our focus is on water-right demand and, in particular, on the type of legal and market mechanisms that may have an incidence on that, so that water use is efficient. In particular we acknowledge the following facts: (1) that many stakeholders are involved in the market; (2) that stakeholders have different and frequently conflicting objectives; (3) that stakeholders have decision-making capabilities; (4) that there is the possibility of establishing conventions that are applicable to the actions of stakeholders, and that stakeholders are capable of complying with those conventions; (5) that these stakeholders are autonomous to comply or not, with the conventions.

With respect to demand we build on two assumptions. First, we assume that water use is granted to individual agents or to groups of individuals through *water rights* that specify the amount of water, period and type of use granted, as well as the location where that water may be extracted. Second, we assume that these rights may be traded.¹

¹The Spanish Water Law and its amendments —'Real Decreto Legislativo 1/200, BOE 176' and the initiatives approved in 2001, 2004 and 2007— institute tradable water rights and the creation of "water banks" during extreme draught conditions.

In brief, we claim that one can see the use of water in a basin as a regulated open MAS and we bet on designing an agent-based market of water rights to micro-model demand and foster efficient use.

We foresee the following potential uses for that market:

A test bed for agreement technologies. From a research perspective, our interest is on the role agreement plays in this social system, on the mechanisms that facilitate an agreement, on the management of agreements, on the normative organizational environments. Thus, we are designing a testbed to provide adequate inspiration for theoretical cogitations on agreement and for the development of the corresponding technologies [3]

A demand component of a sophisticated basin model to visualize and explore water management policies.

A prototype for an online market of water-rights.

II. *mWater*, A REGULATED MAS FOR THE EXCHANGE OF WATER RIGHTS

In this paper we only sketch a bare-bones institutional framework that regiments the market and the main ancillary activities. For the construction of that framework we follow the IIIA *Electronic Institution* (EI) conceptual model [1] where an institutional is specified through two main blocks: one that deals with ontological components (the *dialogical framework* that specifies ontology, language, roles and information model) and another for deontological components (the *performative structure* for interaction models and procedural prescriptions and *rules of behavior* for commitment-making conventions).

We should mention that our framework captures those conventions that are imposed by current legislation and become regimented in the market, but it also captures new conventions to make the market agile and contract management realistic. Thus, for instance, we keep those roles sanctioned by current legislation, but add those that make a richer market or affect conflicts. Likewise, we

keep the (totally ordered) seven types of water use, but specify a water right by a 5-tuple (location, basin district, use, volume, duration), and introduce the possibility of splitting (i.e. trading parts) and joining (i.e. trade a combination) water rights.

The procedural norms in *mWater* are specified through a nested performative structure. The top one, *mWaterPS*, describes the overall market with five active scenes and two sub-structures: *TradingTablesPS* and *AgreementManagementPS*. Interactions in *mWaterPS* start with an *Entitlement* process through which an individual may become a rightful holder of a water right, followed by a process of *Accreditation*, that brings that right into the market. The third scene is a Trading Hall where traders are notified of upcoming negotiations and the reaching of agreements. Actual negotiations take place in the scenes that belong to *TradingTablesPS* and once an agreement on transferring a water right has been reached it is "managed" according to the market conventions captured in *AgreementManagementPS*. Two final scenes take care of the (permanent) annulment and (temporary) suspension of rights. The *TradingTablesPS* includes a scene schema for each trading mechanism. Currently, a right-holder may opt for a standard double auction or a closed bid or face-to-face negotiation but other mechanisms may be added as needed. *AgreementManagementPS* works roughly as follows: First of all, when an agreement is reached, *mWater* staff checks if the agreement satisfies some formal conditions and if so, a transfer contract is signed. When a contract is active, other right-holders and external stakeholders may initiate a grievance procedure that impact the transfer agreement. *AgreementManagementPS* includes different scenes to address such grievances or for the disputes that may arise among co-signers. If things proceed smoothly, the right subsists until maturity.

III. RESEARCH OPPORTUNITIES

mWater allows us to envision the following research opportunities:

Organizational models that are dynamic and flexible enough to specify evolving regulated market scenarios. Because water's unique characteristics, *mWater* requires organization structures that restrict the way agreements are reached by fixing the social structure of the participating entities, the capabilities of their roles and the relationships among them (e.g. power, authority).

Reasoning about normative regulation and social norms for negotiation and execution of agreements and contracts. On and off-line, from an individual agent's perspective and from the market design perspective. Dynamics of norms and norm adoption.

Techniques for flexible on-demand individual and collective negotiation among humans or non-human actors (i.e., agents and services). It is often the case that a water right holder is motivated to achieve a goal (buy or sell a right) that is only possible by gaining the collaboration of others (i.e., a federation of water right holders), then it is required to generate an explicit mutually acceptable agreement through negotiation and to define detailed workflows that regulate the activities and combinations of roles in the organization as well as their associated data flow. *mWater* also requires models and techniques for judgement aggregation, argumentation, persuasion, normative reasoning and agreement planning.

Models for agreement conceptualization and patterns specification, e-Contracting. Relations among different agreements (sub-agreements), for example a situation in which in order to get a water right transfer a buyer requires to contract a transportation resource from other users.

Techniques for initiation, coordination, and supervision of different forms of agreement, contracts and grievances. Even when water right agreements or contracts are signed, the behavior of the participating entities might not be completely determined as their autonomy and selfishness might cause them not to honour their commitments if there is a potential gain in doing so. Online Dispute Resolution environments.

An approach to summarize the life-cycle of agreements in order to build long-term relationships between the water rights market participants.

IV. ACKNOWLEDGEMENT

This paper was partially funded by the Consolider programme of the Spanish Ministry of Science and Innovation through project AT (CSD2007-0022, INGENIO 2010).

REFERENCES

- [1] Josep Arcos, Marc Esteva, Pablo Noriega, Juan Rodriguez-Aguilar, and Carles Sierra. Engineering open environments with electronic institutions. *Engineering Applications of Artificial Intelligence*, (18):191–204, 2005.
- [2] F. Martin de Santa Olalla, A. Dominguez, F. Ortega, A. Artigao, and C. Fabeiro. Bayesian networks in planning a large aquifer in eastern manch, Spain. *Environmental Modelling and Software*, 22:1089–1100, 2007.
- [3] Juan A. Rodriguez-Aguilar. AGREEMENT TECHNOLOGIES. Towards a new programming paradigm for agent-oriented technologies. *Position Statement at Future of Software Engineering and Multi-Agent Systems - FOSE-MAS 2008*, 2008.
- [4] M. Thobani. Formal water markets: Why, when and how to introduce tradable water rights. *The World Bank Research Observer*, 12(2):161–179, 1997.

Open Interaction System Specification and Monitoring Using Semantic Web Technology

Nicoletta Fornara Università della Svizzera italiana, via G. Buffi 13, 6900 Lugano, Switzerland
Email: nicoletta.fornara@usi.ch

I. INTRODUCTION

The design and development of *open distributed interaction systems*, where heterogeneous, autonomous, and self-interested agents can interact by entering and leaving dynamically the system, is widely recognized to be a crucial issue in the development of nowadays applications on the Internet: like e-commerce applications [1], collaborative social systems [2], or application to support the automatic management of virtual organizations [3]. In particular in our view the interacting agents may range from very complex autonomous software agents able to reason and to plan their actions and that behave on behalf of their human owners, to very simple software used by human beings as an interface to interact with the system.

Given that the agents are assumed to be heterogeneous because they may be developed by different designers or they may be human beings, no assumptions can be made on their internal architecture. Given that the system is open and agents may enter and leave it dynamically, it is necessary to find a standard way for specifying a communication language for the interacting agents and for defining the context and the rules of the interaction. Moreover given that the interacting agents are autonomous it is necessary to find a way to regulate interactions so that agents may have reliable expectations on the future development of the system. Furthermore given that these systems will be used to enrich and improve human beings interactions, it is crucial that the proposed design approach is defined taking inspiration from existing studies about human interactions.

Starting from these requirements in our previous works [4], [5], [6], [7], [8] we proposed a meta-model for the conceptual design of open interaction systems based on speech act theory [9], [10] and on Searle's theory on construction of social reality [11]. In particular we proposed the *OCeAN* metamodel, which is based on the definition of a set of application independent concepts that have to be used in the specification of every type of interaction system. We initially proposed an agent communicative language whose semantics is based on the notion of *social commitment* and *temporal proposition*.

Therefore in order to be able to define the semantics of declarative communicative acts, we introduced in our model other institutional concepts, like the notion of *institutional action*, *institutional power*, and *role*. Finally in order to constrain agents' actions we formalized the notion of *norm* and of *sanction* or *reward* that are used for norm enforcement [12]. We model open interaction systems as a set of *artificial institutions*. In particular in our view the definition of a specific artificial institution consists of: (i) a component, called meta-model, which includes the definition of basic entities common to the specification of every institution, like the concepts of commitment, institutional power, role, and norm, and the actions necessary for exchanging messages; (ii) a component specific to the institution in question, which includes the specification of the powers and norms that apply to the agents playing roles in the institution, and the definition of the concepts pertaining to the domain of the interaction (for example the actions of paying or delivering a product, bidding in an auction, etc.).

Regarding the language used to specify the various components of the model we initially adopted a language with an operational intuitive semantics based on the notion of object and attribute close to object oriented programming. The difficulty that we experimented with this approach was in developing agents able to reason on their actions and able to monitor the agent's behavior. We therefore proposed a formalization of the *OCeAN* meta-model based on the Discrete Event Calculus [8]. This approach resulted very fruitful for unambiguously specifying the concepts of our meta-model and for being used to simulate the time evolution of an actual interaction, but we experimented performance problems and we did not find a simple way to interface our event calculus specification with an external application used to enable agents interactions, like for instance the *JADE* framework¹.

We therefore decide to follow a new approach that we plan to investigate and evaluate in the future. Our idea is to use standard Semantic Web Technology to specify,

¹<http://jade.tilab.com/>

to reason on, and to monitor agent's actions. In [13] we started to formalize the deontic part of the *OCeAN* meta-model using OWL 2 DL², SWRL rules (Semantic Web Rule Language³), and a Java application, developed using OWL-API⁴ and the source code of the Pellet⁵ reasoner, to overcome certain expressiveness limitations of OWL. More precisely, we showed how it is possible to specify and monitor the time evolution of social commitment used to express conditioned obligations and prohibitions on time intervals.

The main advantages of using a decidable logical language like OWL to specify an open interaction system are that Semantic Web technologies are increasingly becoming a standard for Internet applications and therefore they are supported by many reasoners (like Fact++⁶, Pellet⁷, or Racer⁸); moreover ontologies and reasoning services are easily interfaced with applications programmed in Java or other well known languages.

When facing this approach we discovered that there may be the following main problems. Firstly the treatment of time: OWL has no temporal operators, in some cases it is possible to bypass the problem by using SWRL rules and built-ins for comparisons, but in any case this does not provide full temporal reasoning capabilities; notice furthermore that using the OWL Time Ontology⁹ would not be a solution, given that its axiomatization is very weak. Secondly the open-world assumption: in many applications nor being able to infer that an action has been performed is sufficient evidence that the action has not been performed. We faced this second problem by using an external program to simulate a closed world assumption by adding certain closure axioms to the ontology. But there is still the open problem of understanding what part of the model it is better and possible to represent in the ontology in order to be able to reason on it and what part of the model it is better to represent in the external application because current semantic web standards do not support its representation.

REFERENCES

- [1] M. Esteva, D. de la Cruz, and C. Sierra, "Islander: an electronic institutions editor," in *Proceedings of the First International*

²http://www.w3.org/2007/OWL/wiki/OWL_Working_Group

³<http://www.w3.org/Submission/SWRL/>

⁴<http://owlapi.sourceforge.net/>

⁵<http://clarkparsia.com/pellet>

⁶<http://owl.man.ac.uk/factplusplus/>

⁷<http://clarkparsia.com/pellet>

⁸<http://www.sts.tu-harburg.de/~r.f.moeller/racer/>

⁹<http://www.w3.org/TR/owl-time/>, <http://www.w3.org/2006/time.rdf>

- Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2002)*, C. Castelfranchi and W. L. Johnson, Eds. ACM Press, 2002, pp. 1045–1052.
- [2] V. Dignum, F. Dignum, and J.-J. Meyer, "An agent-mediated approach to the support of knowledge sharing in organizations," *Knowl. Eng. Rev.*, vol. 19, no. 2, pp. 147–174, 2004.
- [3] H. L. Cardoso and E. Oliveira, "Virtual enterprise normative framework within electronic institutions," in *Engineering Societies in the Agents World V*. Springer, 2004, pp. 14–32.
- [4] N. Fornara and M. Colombetti, "Operational specification of a commitment-based agent communication language," in *Proceedings of the First International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2002)*, C. Castelfranchi and W. L. Johnson, Eds. ACM Press, 2002, pp. 535–542.
- [5] —, "Defining interaction protocols using a commitmentbased agent communication language," in *Proceedings of the Second International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2003)*, J. Rosenschein, T. Sandholm, M. Wooldridge, and M. Yokoo, Eds. ACM Press, 2003, pp. 520–527.
- [6] N. Fornara, F. Viganò, and M. Colombetti, "Agent communication and artificial institutions," *Autonomous Agents and Multi-Agent Systems*, vol. 14, no. 2, pp. 121–142, April 2007.
- [7] N. Fornara, F. Viganò, M. Verdicchio, and M. Colombetti, "Artificial institutions: A model of institutional reality for open multiagent systems," *Artificial Intelligence and Law*, vol. 16, no. 1, pp. 89–105, March 2008.
- [8] N. Fornara and M. Colombetti, *Specifying Artificial Institutions in the Event Calculus*, ser. Information science reference. IGI Global, 2009, vol. Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models, ch. XIV, pp. 335–366.
- [9] J. L. Austin, *How to Do Things With Words*. Oxford: Oxford University Press, 1962.
- [10] J. R. Searle, *Speech Acts: An Essay in the Philosophy of Language*. Cambridge, United Kingdom: Cambridge University Press, 1969.
- [11] —, *The construction of social reality*. New York: Free Press, 1995.
- [12] N. Fornara and M. Colombetti, "Specifying and enforcing norms in artificial institutions," in *Declarative Agent Languages and Technologies VI 6th International Workshop, DALT 2008, Revised Selected and Invited Papers*, ser. LNCS, M. Baldoni, T. Son, B. van Riemsdijk, and M. Winikoff, Eds., vol. 5397. Springer Berlin / Heidelberg, 2009, pp. 1–17.
- [13] —, "Ontology and time evolution of obligations and prohibitions using semantic web technology," in *Proceedings of the Workshop on Declarative Agent Languages and Technologies DALT@AAMAS09, Budapest, Hungary, 11 May 2009*, M. Baldoni, J. Bentahar, and M. v. R. J. Lloyd, Eds., 2009.

The SIOC Project: Semantically-Interlinked Online Communities

Uldis Bojārs and Alexandre Passant and John G. Breslin and Stefan Decker

Digital Enterprise Research Institute,

National University of Ireland, Galway, Ireland.

Email: [uldis.bojars,alexandre.passant,john.breslin,stefan.decker]@deri.org

Abstract—The SIOC project is aimed at expressing information about the structure and content of online community sites and at enabling interoperability on the Social Web using Semantic Web technologies. In this paper we briefly describe the SIOC project, introduce the SIOC Core ontology and its modules, and discuss some SIOC-based applications in terms of human and agent communication.

I. INTRODUCTION

While new paradigms, tools and web services introduced by the Social Web (such as blogs, wikis, tagging practices) are now widely accepted in both public and scientific communities, these tools generally act as independent data silos; hence, interoperability between applications is a complex issue. The SIOC project¹ aims at solving this by providing a comprehensive data model (as well as related tools and applications) based on Semantic Web technologies [1] in order to represent online communities and their activities in an homogenous way.

The SIOC project consists of two main parts:

- The SIOC Ontology, composed of a Core ontology and different modules;
- A set of applications, for both producing and consuming SIOC data, constantly evolving based on the user needs and implementations.

II. THE SIOC ONTOLOGY

The SIOC Ontology is composed of a Core ontology and a set of modules, focusing on the ease of integration of SIOC in existing applications by Web developers. With an emphasis on standard Semantic Web technologies since its beginning, the whole ontology has been designed using RDF(S)/OWL. A comprehensive overview of the SIOC ontology is provided in [2].

A. SIOC Core Ontology

Main classes and properties in the SIOC ontology² are shown in Figure 1. While relatively small and simple, this model is yet powerful enough to represent the content produced and exchanged within online communities. For instance, a `Forum` represents a space in which discussion happen (not necessarily a bulletin board, in spite of its name, but for example as a weblog), and contains different `Posts`, written

by `Users`. The following example then describes that Alice has created a post in a particular forum (*i.e.* an area of discussion) and that Bob replied to it — as follows (using the N3 notation - prefixes omitted). In order to represent more abstract containers (such as a personal information space), more general `Container` and `Space` classes can be used.

```

:post a sioc:Post ;
  sioc:has_creator :alice ;
  sioc:has_container :forum ;
  sioc:has_reply :reply .
:forum a sioc:Forum .
:reply a sioc:Post ;
  sioc:has_creator :bob .
:alice a sioc:User .
:bob a sioc:User .

```

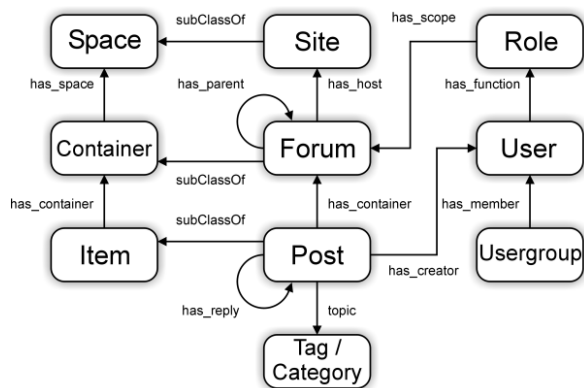


Fig. 1. Main classes and properties in the SIOC ontology.

Other properties and classes exist in the ontology, for instance properties to represent previous and next versions of an item, which can be used when representing Wikis.

It is worth noticing that each `sioc:User` is actually related to the `Agent` class (from the FOAF³ — Friend Of A Friend [3] — ontology), and that `sioc:User` can be consequently associated with both software agents and human users. Hence, SIOC can be used to represent communication between bots on the Social Web (*e.g.* on IRC using the

¹<http://sioc-project.org>

²<http://rdfs.org/sioc/spec>

³<http://foaf-project.org>

SIOC-IRCllog project⁴) and then enable a machine-readable description of communities of agents interacting together on social websites. Moreover, another link between FOAF and SIOC is that the social networking aspect can be represented using FOAF (by `foaf:knows`), while SIOC can be used to represent aspects of relationships inside the community, e.g. a user following another (on microblogging services) can be expressed by the `sioic:follows` property.

B. SIOC Modules

Several SIOC modules have been defined to extend the available terms and to avoid making the SIOC Core Ontology too complex and unreadable. SIOC has 4 ontology modules: Access, Argumentation, Services and Types⁵.

- the Access module defines simple classes and properties regarding the notions of Role and Permission to represent access rights and permissions in online communities websites;
- the Argument module defines classes and properties to represent simple argumentative discussions in online communities websites.
- the Services module defines classes and properties to represent Web services related to online communities (e.g. API endpoint and return format, etc.);
- the Types module defines advanced content-types to be used when defining user-generated content from online-communities. For instance, it includes classes such as `sioic:BlogPostor` `sioic:Wiki` that respectively subclass the `sioic:Postand` `sioic:Forum` classes from the Core Ontology;

In addition, one of the recent development of SIOC is a module defining alignments between SIOC and the SWAN — Semantic Web Applications in Neuromedicine — ontology [4]⁶, providing a complete model for fine-grained argumentative discussions in online scientific communities.

III. STATUS AND UPDATE OF SIOC

Since the goal of SIOC is to provide interoperability between communities on the Social Web, one way to evaluate its success is to consider its uptake on the Web. To illustrate the amount of SIOC data on the Web, according to the PingTheSemanticWeb (PTSW) service⁷ on June 2009 there were 132'475 URIs which contain data described using the SIOC ontology⁸.

In addition to the uptake in terms of number of documents, SIOC is now widely accepted as a core ontology to describe Social Web communities using Semantic Web technologies, alongside with FOAF. Hence, the use of SIOC is suggested by the Yahoo! SearchMonkey developer documentation⁹ (SIOC

data being indexed by SearchMonkey to improve presentation of search results) and by various best practices documents describing data publishing on the Semantic Web such as [5].

Various SIOC-enabled applications have been created¹⁰, forming a *food-chain* (Figure 2) that includes data exporters for Web 2.0 platforms and services (e.g. Drupal¹¹, WordPress, Flickr, etc.), tools for indexing of distributed data, and browser and visualization applications, allowing users to consume SIOC data from various sources in an integrated way.

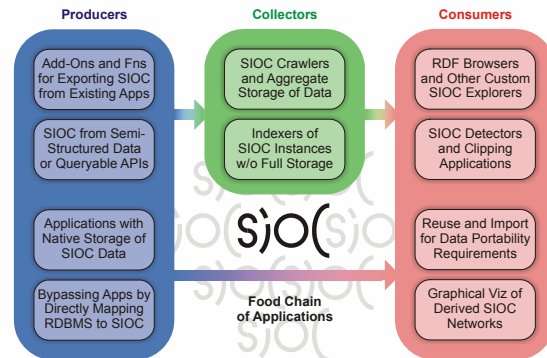


Fig. 2. The SIOC food-chain

IV. CONCLUSION

In this paper, we briefly introduced the SIOC project, its goals and means as well as uptake and services overview. In particular, in the context of multi-agent systems and online communities, an already active fields for several years if we consider for instance [6], the SIOC ontology could be used to express in a unified way information about activities of both humans and agents interacting in these communities.

ACKNOWLEDGEMENTS

The work presented in this paper has been funded in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Líon-2).

REFERENCES

- [1] T. Berners-Lee, J. A. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, vol. 284, no. 5, pp. 34–43, 2001.
- [2] U. Bojars, J. G. Breslin, V. Peristeras, G. Tummarello, and S. Decker, "Interlinking the Social Web with Semantics," *IEEE Intelligent Systems*, vol. 23, no. 3, pp. 29–40, May/June 2008.
- [3] D. Brickley and L. Miller, "FOAF Vocabulary Specification," FOAF Project, Namespace Document 2 Sept 2004, 2004, <http://xmlns.com/foaf/0.1/>. [Online]. Available: <http://xmlns.com/foaf/0.1/>
- [4] P. Ciccarese, E. Wu, G. Wong, M. Ocana, J. Kinoshita, A. Ruttenberg, and T. Clark, "The SWAN biomedical discourse ontology," *Journal of Biomedical Informatics*, vol. 41, no. 5, pp. 739–751, 2008.
- [5] C. Bizer, R. Cyganiak, and T. Heath, "How to Publish Linked Data on the Web," Tech. Rep., 2007, <http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/>.
- [6] F. Gandon and R. Dieng-Kuntz, "Distributed Artificial Intelligence for Distributed Corporate Knowledge Management," in *CIA 2002 - 6th Cooperative Information Agents - Intelligent Agents for the Internet and Web September*. Springer LNAI 2446, pp. 202–217.

¹⁰<http://rdfs.org/sioic/applications/>

¹¹<http://drupal.org/project/sioic>

⁴<http://irc.sioic-project.org/about.html>

⁵Information about SIOC modules: <http://rdfs.org/sioic/spec/#sec-modules>

⁶<http://rdfs.org/sioic/swan>

⁷<http://pingthesemanticweb.com>

⁸The full amount of SIOC information on the Web is larger than described here as PTSW indexes only a part of available RDF data.

⁹http://developer.yahoo.com/searchmonkey/smguides/profile_vocab.html

Norms, Organisations and Semantic Web Services: The ALIVE approach

Sergio Alvarez-Napagao*, Owen Cliffe†, Javier Vazquez-Salceda* and Julian Padget†

*Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Barcelona, ES

†Department of Computer Science, University of Bath, BATH BA2 7AY, UK

Abstract—ALIVE is an EU FP7 STREP whose goal is the convergence of organisational and normative modelling with and service-oriented architectures (SOAs) using model-driven software engineering. The project provides a framework for designing and implementing systems, taking into account organisational, coordination and service perspectives. A key project aspect is the integration of normative systems with live SOAs, through the distributed monitoring of normative state. Here we give a brief overview of the project, explore of the domain from a service context, outline the architecture under construction and sketch the use-cases that illustrate and inform the project.

I. INTRODUCTION

Recent developments in service-oriented architectures, including the proliferation of web services (both in a conventional context and in the context of the semantic web) have opened up the possibility for a new class of loosely coupled, open system. The software engineering methodology employed however constrains the resulting system and approaches such as UML lack the means both to capture high-level, abstract whole-system properties and to maintain them at run time of systems without explicit (user) specification.

In the context of socially-situated systems organisational models provide a way to model the relationships between users and system components as roles. In addition norms allow designers to specify succinctly the regulations that govern actors playing given roles and the expectations on their behaviour. These relationships may be maintained as the normative state of a system either explicitly or implicitly at run time, allowing designers to identify quickly where given system behaviours (specified as norms) are being upheld or violated. The ALIVE project aims to link existing work in modelling organisational structures and norms with the state of the art in service-oriented computing, allowing designers to build service oriented architectures that enact organisational interactions in a natural way. The process of engineering the systems themselves remains very complex, so the project takes advantage of model-driven development methodology to assist in the development of tools and software by offering the potential for verifiable automation of key translation processes.

II. THE ALIVE CONTEXT

The ALIVE architecture combines *model driven development* (MDD) [6] with coordination and organisational mechanisms, providing support for *live* and *open* systems of services. ALIVE's approach extends current trends in engineering by

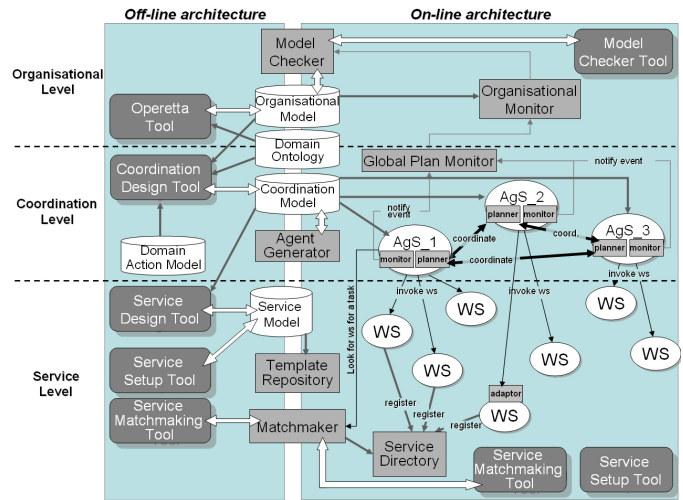


Fig. 1. ALIVE Multi-Level Architecture

defining three levels in the design and management of distributed systems: the Service, Coordination and Organisation Levels, illustrated in Fig. 1, and explained below.

The *Service Level* extends existing service models, to make components aware of their social context and of the rules of engagement with other components and services, by means of semantic Web technologies. This “semantification” is particularly useful when highly dynamic and frequently changing services (the WSs in Fig. 1) are present, as the meta-information in each service description (stored in a *service directory*) aids tasks such as finding substitute services (either via a matchmaker or manually) when the original fails.

The *Coordination Level* provides the means to specify, at a high level, the patterns of interaction among services, using a variety of powerful coordination techniques from recent agent research [2], [5]. These are represented by *agentified services*, that are organisationally-aware, meaning they are aware of system objectives and manage task allocation and workflow generation and agreement. Also, at the coordination level agreed workflows can be adapted *while* the system is running—essential when the system has to react to failures or exceptions (e.g., failing payment or booking systems).

The *Organisational Level* provides a social context for the Coordination and Service levels, specifying the organisational rules that govern interaction and using recent developments in organisational dynamics [7] to allow the structural adaptation

of systems over time. This is important when frequent changes of rules and restrictions are expected.

The ALIVE architecture can be seen as a service-oriented middleware supporting the combination, reorganisation and adaptation of services at both design- and run-time. These activities follow organisational patterns and adopt coordination techniques. Furthermore, the MDD paradigm offers significant developer assistance, through semi-automated transformations between models of the three levels, as well as the capacity for multiple target platforms and representation languages. More details of the theoretical and methodological aspects are available as public deliverables on the ALIVE website (<http://www.ist-alive.eu>).

III. PROJECT USE-CASES

We now outline the three use cases on which we draw to demonstrate and evaluate the effectiveness of the approach.

Intelligent tourism: The first use case (from TMT Factory (<http://tmtfactory.com>), a company selling smart displays for urban tourist environments) focuses on providing assistance and advice to tourists through a variety of devices (including static displays). At the organisational level the use case applies models of city, regional and national laws to tune the types of services which are offered to customers (for instance night clubs not being recommended to under 18s), at the coordination it negotiates connected services (such as transport and activities) and at the service level it interacts with existing providers (such as cinema booking services and transport planning services).

Disaster Management: The second use case (from Thales D-CIS laboratory (<http://www.decis.nl/>)) extends existing work on the modelling and simulation of urban disasters (here the flooding of the Rotterdam harbour area). At an organisational level we model the structure and roles of the services (e.g. city councils, local and national disaster centres and emergency services) involved in handling a disaster at various scales (from simple localised flooding to a city-wide flood) and the regulations relating to how those parties interact. At the coordination level we model interactions between parties, instructions to particular agents (e.g. dispatching fire trucks to a particular area) and at the service level we link to an existing simulation environment which models the physical conditions of a flood as it happens.

Communication in Virtual Environments: The final use case (from Calico Jack Ltd. (<http://www.calicojack.co.uk/>), a networking research and services provider) deals with modelling human norms and conventions relating to conversation in virtual and on-line environments. The use case assumes that users are active in a number of contexts (such as FaceBook, Second life, Bebo, via SMS and over email) and handles the intelligent redirection of communication between users within these contexts, preserving modelled norms relating to those communications, such as a user's desire not to receive unsolicited communication from unknown parties while in Second Life. At the coordination level models relationships between users and their presence in particular contexts, while

at the service level the focus is on low-level interaction with the underlying services (e.g. via game APIs, or SMS gateways) and composing services to interchange information contained within communications, such as transcribing voice-mail messages to text for delivery within a text-only environment.

IV. CURRENT STATUS

The key to the success of the project is integrating state-of-the-art approaches and tools from each of the areas of interest (organisations, coordination and services). To date, the project has focused on developing an end-to-end methodology and meta-model which unifies and relates concepts as follows: (i) Using the OPERA [1] organisational model and corresponding Operetta tool organisational concepts (Roles, Interactions, and Norms) can be specified. (ii) These concepts are reified as coordination actions and agents, which in turn may be used to build coordination plans (using the GPGP/TÆMS coordination framework [3]) for groups of agents enacting roles within the organisation. (iii) Actions in coordination plans link to the provision or consumption of semantic web services, by matchmaking tasks to existing semantic service descriptions and services using a hybrid matchmaker derived from OWL-S MX [4]. We are developing tools to assist in the process of semantic annotation of services. (iv) Agents enact their roles within the (either autonomously or via human controllers) organisation, by direct communication or by invoking the selected services, maintaining relevant state. (v) Monitors observe agent interactions, normative and organisational state (e.g. the status of agents' obligations, permissions, powers and the roles currently being assumed), and the agents interact with these monitors allowing them reason about the normative effects of their actions and re-plan after service failures.

Planned project deliverables include a comprehensive development methodology and a suite of Eclipse-based development tools (design and run time user assistance tools and programming libraries for system development).

Acknowledgement: This work has been supported in part by the European Commission, project FP7-215890 (ALIVE).

REFERENCES

- [1] Dignum, V.: A Model for Organizational Interaction Based on Agents, Founded in Logic. PhD thesis, University of Utrecht. 2004.
- [2] Ghijssen, M., Jansweijer, W., Wielinga, B.B.: Towards a Framework for Agent Coordination and Reorganization, AgentCoRe. In: Coordination, Organizations, Institutions, and Norms in Agent Systems III. LNCS, vol. 4870, pp. 1–14. Springer, Heidelberg (2008).
- [3] Lesser, V.: Evolution of the GPGP/TAEMS domain-independent coordination framework. In AAMAS '02: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, pages 1-2, New York, NY, USA, 2002. ACM.
- [4] Klusch, M., Fries, B., Sycara, K., Automated Semantic Web Service Discovery with OWLS-MX, Proc. 5th Intl Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS), Hakodate, Japan, 2006.
- [5] Matskin, M. et al.: Enabling Web Services Composition with Software Agents. In: Proc. of the Conference on Internet and Multimedia Systems, and Applications. Honolulu (2005)
- [6] OMG: Model Driven Architecture. <http://www.omg.org/mda/>. Retrieved 20090527.
- [7] van der Vecht, B., Dignum, F., Jules, J., Meyer, Ch., Dignum, V.: Organizations and Autonomous Agents: Bottom-up Dynamics of Coordination Mechanisms. In: 5th Workshop on Coordination, Organizations, Institutions, and Norms in Agent Systems. Estoril (2008).

Introduction to the Proceedings of FAMAS'09 Formal Approaches to Multiagent Systems

(Introductory Essay of the Workshop)

Barbara Dunin-Kępicz* and Rineke Verbrugge†

*Institute of Informatics, Warsaw University
Banacha 2, 02-097 Warsaw, Poland and

Institute of Computer Science, Polish Academy of Sciences
Ordona 21, 01-237 Warsaw, Poland Email: {kepicz}@mimuw.edu.pl

†Institute of Artificial Intelligence, University of Groningen
PO Box 407, 9700 AK Groningen, The Netherlands
Email: rineke@ai.rug.nl

Abstract

In recent years, multiagent systems have come to form one of the key technologies for software development. The Formal Approaches to Multiagent Systems (FAMAS) workshop series brings together researchers from the fields of logic, theoretical computer science and multi-agent systems in order to discuss formal techniques for specifying and verifying multiagent systems, including many subtle and not easy to formalize aspects of agency.

FAMAS addresses logics and formal methods for multiagent systems. Specifically, the workshop series addresses formal approaches to cooperation, multi-agent planning, communication, coordination, negotiation, games, and reasoning under uncertainty in a distributed environment.

FORMAL APPROACHES TO MULTIAGENT SYSTEMS: THE FOUR WORKSHOPS

The first FAMAS workshop, FAMAS'03, was a successful satellite event of the European Conference on Theory and Practice of Software (ETAPS'03) in Warsaw. It took place on April 12th 2003, and afterwards a selection of contributed and invited papers was published in *Fundamenta Informaticae* as volume 63, issue 2,3 of 2004.

The second FAMAS workshop, FAMAS'06, took place on Monday 28 August, 2006 in conjunction with the European Conference on Artificial Intelligence (ECAI'06) at the Riva del Garda. Again, a selection of FAMAS speakers were invited to contribute an extended version of their work to a special issue of a well-known international journal, this time the *Journal of Autonomous Agents and Multiagent Systems* (JAAMAS), volume 19 (1), 2009.

The third FAMAS workshop, FAMAS'007, was one of the agent workshops gathered together under the umbrella of MALLOW'007 and AGENTS'007, taking place from 3 to 7 September 2007 in Durham. A special issue of the *Logic Journal of the IGPL* will gather extended versions of the best papers of FAMAS'007.

As its predecessor, this fourth FAMAS workshop, FAMAS'09, is a part of MALLOW, this time organized in Torino, from September 7 to 10, 2009. We look forward to a lively workshop with high-quality contributions. Again, best papers will be invited to submit an extended version for a special issue of the *Logic Journal of the IGPL*.

Let us give a short preview of the volume. All research reported here is squarely related to practice, even if the formal approach is taken. Thus, just as in previous installments of FAMAS, contributions devote their attention to pressing practical problems such as supporting organizations, allocating goods in a fair way, and effective communication.

Since the first FAMAS edition, emphasis has been shifting to correspond to multi-agent systems being situated in a dynamic environment. Also, quite a few authors take on the challenge to combine different logics or to investigate the relations between different formal viewpoints, for example game theory and learning theory, or epistemic logic and belief revision, in a methodologically sound manner.

MECHANISMS FOR REACHING AGREEMENT

Gujar and Narahari, in their paper “Redistribution mechanisms for the assignment of heterogeneous objects”, consider a problem of which one can meet many variations in practice. Suppose that there is a certain number of different resources available, and that there is a higher number of agents, each of them interested in using one of the resources.

It seems clear that one should try to assign the resources in such a way that the agent who values an object the most gets it, and preferably in such a way that in the end the total transfer of money in the system is balanced: the system or the auctioneer are not left with a deficit or a surplus. This may be done by redistributing some money after the allocation of the objects to the paying agents. The authors show which types of constraints on the redistribution mechanism are feasible, and which are not.

In “Talking your way into agreement: Belief merge by persuasive communication”, Baltag and Smets take a dynamic approach to the issue of how a group of agents’ individual knowledge and belief could be merged into a single set of group beliefs. The simplest example is that of distributed ‘hard’ knowledge, which can be transformed into group knowledge if the agents make truthful public announcements of all their individual knowledge. At the other extreme, for agents’ ‘soft’ beliefs, one could enforce a hierarchy by giving priority to certain agents’ beliefs over others in a so-called lexicographic merge.

The authors also consider some more or less democratic intermediate possibilities, such as sincere persuasive public announcements of ‘soft’ (defeasible) knowledge. They illustrate their definitions with concrete examples. It turns out the essential role of the person who sets the agenda and determines the order in which speakers and issues are scheduled.

NEW APPROACHES TO KNOWLEDGE AND BELIEF

In the paper “Comparing strengths of beliefs explicitly”, authors de Jongh and Ghosh introduce a novel ordering of formulas: $\varphi \geq_B \psi$ for an agent, if its strength of belief in φ is greater than that in ψ . Such explicit comparative formulas can be used to express notions such as plausibility and disbelief in φ , where an agent’s degree of belief in $\neg\varphi$ is greater than that in φ .

The authors show how this added expressivity to the standard logic of beliefs helps to model common decision situations. The authors investigate both the resulting comparative belief ordering and a different but related plausibility ordering reminiscent of David Lewis’ sphere systems for counterfactuals. Then they provide a sound and complete axiomatization for the single-agent case. Finally, they extend their approach to multi-agent situations and dynamic environments.

Schwarzentruber aims to develop a multi-agent epistemic logic based on spatial geometric semantics in his contribution “Knowledge about lights along a line”. Interestingly, the paper has been inspired by didactical considerations: in a given concrete situation in which agents and lamps are placed on a line, what do agents know about lamps and about the knowledge of the other agents of these? Such concrete reasoning based on what agents can and cannot see, may help students to learn abstract aspects of epistemic logics, such as higher-order knowledge and the effects of public announcements.

STRATEGIC GAMES

Gierasimczuk, Kurzen and Velazquez-Quesada, in their paper “Games for learning: a sabotage approach”, also cast their results in an educative framework, in their case in the much more abstract setting of learning theory. They introduce calibrated versions of a sabotage game, where a ‘teacher’ is sometimes helpful to a ‘student’ in getting from the initial state to a final state, sometimes not, and where also the student may display different degrees of helpfulness.

The paper describes the interactive nature of learning by game-theoretical and logical means. The existence of a winning strategy is characterized by formulas in Sabotage Modal Logic, and the paper is rounded off with a complexity analysis.

The contribution by Chandrashekar and Narahari, “On the incentive compatible core of a procurement network game with incomplete information” is also set squarely in a game-theoretical framework, namely that of cooperative games with incomplete information. Suppose that a single buyer is interested in buying multiple units of a single item, that can be supplied through a linear supply chain in which multiple suppliers may take part. At each stage, all suppliers have their own costs and a limited capacity . How can this negotiation problem be solved in a manner profitable to all participants if the information such as each supplier’s cost is only privately held? Taking off from earlier work by Myerson, the authors show precisely how such allocations to coalitions of suppliers can be done.

FORMAL APPROACHES TO THE DYNAMICS OF MULTI-AGENT SYSTEMS

Goranko and Shkatov, in their contribution “Tableau-based decision procedure for the full coalitional multiagent logic of branching time”, take a second look at the coalitional multi-agent temporal-epistemic logic of branching time, as introduced by Halpern and Vardi in 1989. The new logic extends standard computation tree logic (CTL) by adding modalities expressing distributed and common knowledge among all possible teams of agents in the language.

Goranko and Shkatov provide an elegant tableau-based decision procedure for this logic. They prove that the procedure is sound and complete, and show that the decision procedure works in exponential time. Therefore in terms of complexity, their quite expressive logic fares no worse than standard epistemic logic with common knowledge.

In their paper “A framework to model norm dynamics in answer set programming”, the authors Panagiotidi, Nieves and Vazquez-Salceda fruitfully apply methods from logic programming to the representation of deontic concepts like prohibitions, obligations and permissions in normative multi-agent systems.

Their Answer Set Programming approach provides a neat operational semantics that helps to model how agents reason about norms and actions in a dynamic setting. Thus, important aspects like effects of actions, deadlines, violations and sanctions are naturally represented. The authors present some results about reachability and a partial implementation.

ACKNOWLEDGMENTS

We would like to thank all the people who helped to bring about FAMAS'09. First of all, we thank all speakers for ensuring a diverse and interesting workshop. Special thanks are due to the members of the program committee for their professionalism and their dedication to select papers of quality and to provide authors with useful, constructive feedback during the in-depth reviewing process:

Program committee

- Hans van Ditmarsch, University of Aberdeen, UK and University of Sevilla, Spain
- Andreas Herzig, Université Paul Sabatier, France
- Wiebe van der Hoek, University of Liverpool, UK
- Wojtek Jamroga, Clausthal University of Technology, Germany and University of Luxembourg
- John-Jules Meyer, Utrecht University, The Netherlands
- Wojciech Penczek, University of Podlasie and Polish Academy of Sciences, Poland
- David Robertson, University of Edinburgh, UK
- Jeff Rosenschein, The Hebrew University of Jerusalem, Israel
- Carles Sierra, IIIA, Spanish Research Council, Spain
- Andrej Szalas, Warsaw University, Poland and Linköping University, Sweden
- Onn Shehory, IBM - Haifa Research Labs, Israel

The FAMAS website would not exist without the help of Michal Ślizak in Warsaw.

Warszawa
Groningen

Barbara Dunin-Kępicz
Rineke Verbrugge

August 2009

Redistribution Mechanisms for Assignment of Heterogeneous Objects

Sujit Gujar

Dept of Computer Science and Automation
Indian Institute of Science
Bangalore, India
sujit@csa.iisc.ernet.in

Y Narahari

Dept of Computer Science and Automation
Indian Institute of Science
Bangalore, India
hari@csa.iisc.ernet.in

Abstract

There are p heterogeneous objects to be assigned to n competing agents ($n > p$) each with unit demand. It is required to design a Groves mechanism for this assignment problem satisfying weak budget balance, individual rationality, and minimizing the budget imbalance. This calls for designing an appropriate rebate function. Our main result is an impossibility theorem which rules out linear rebate functions with non-zero efficiency in heterogeneous object assignment. Motivated by this theorem, we explore two approaches to get around this impossibility. In the first approach, we show that linear rebate functions with non-zero efficiency are possible when the valuations for the objects have some relationship. In the second approach, we show that rebate functions with non-zero efficiency are possible if linearity is relaxed.

Keywords: Groves Mechanism, Budget imbalance, Redistribution mechanism, Rebate function

1. Introduction

Consider that p resources are available and each of $n > p$ agents is interested in utilizing one of them. Naturally, we should assign these resource such that those who value them most get it. Since Groves mechanisms [13], [3], [6] have attractive game theoretic properties such as dominant strategy incentive compatibility (DSIC) and allocative efficiency (AE), Groves mechanisms are widely used in practice. However, in general, a Groves mechanism need not be budget balanced. That is, the total transfer of money in the system may not be zero. So the system will be left with a surplus or deficit. Using Clarke's mechanism [3], we can ensure under fairly weak conditions that there is no deficit of money, that is the mechanism is weakly budget balanced. In such a case, the system or the auctioneer will be left with some money.

Often, the surplus money is not really needed in many social settings such as allocations by the Government among its departments, etc. Since strict budget balance cannot coexist with DSIC and AE (Green-Laffont theorem [5]), we would like to redistribute the surplus to the participants as

far as possible, preserving DSIC and AE. This idea was originally proposed by Laffont [11]. The total payment made by the mechanism as a redistribution will be referred to as the *rebate* to the agents.

In this paper, we consider the following problem. There are n agents and p heterogeneous objects ($n \geq p > 1$). Each agent desires one object out of these p objects. Each agent's valuation for any of the objects is independent of his valuations for the other objects. Valuations of the different agents are also mutually independent. Our goal is to design a mechanism for assignment of the p objects among the n agents which is allocatively efficient, dominant strategy incentive compatible, and maximizes the rebate (which is equivalent to minimizing the budget imbalance). In addition, we would like the mechanism to satisfy feasibility and individual rationality. Thus, we seek to design a Groves mechanism for assigning p heterogeneous objects among n agents satisfying:

- 1) Feasibility (F) or weak budget balance. That is, the total payment to the agents should be less than or equal to the total received payment.
- 2) Individual Rationality (IR), which means that each agent's utility by participating in the mechanism should be non-negative.
- 3) Minimizes budget imbalance.

We call such a mechanism *Groves redistribution mechanism* or simply *redistribution mechanism*. Designing a redistribution mechanism involves design of an appropriate *rebate function*. If in a redistribution mechanism, the rebate function for each agent is a linear function of the valuations of the remaining agents, we refer to such a mechanism as a *linear redistribution mechanism* (LRM). In many situations, design of an appropriate LRM turns out to be a problem of solving a linear program.

Due to the Green-Laffont theorem [5], we cannot guarantee 100% redistribution at all type profiles. So a performance index for the redistribution mechanism would be the worst case redistribution. That is, the fraction of the surplus which is guaranteed to be redistributed irrespective of the bid profiles. This fraction will be referred to as *efficiency* in the rest of the paper (*Note:* This efficiency is not to be

confused with allocative efficiency). The advantage of worst case analysis is that, it does not require any distributional information on the type sets of the agents. It is desirable that the rebate function is deterministic and anonymous. A rebate function is said to be anonymous if two agents having the same bids get the same rebate. So, the aim is to design an anonymous, deterministic rebate function which maximizes the efficiency and satisfies feasibility and individual rationality.

Our paper seeks to non-trivially extend the results of Moulin [12] and Guo and Conitzer [8] who have independently designed a Groves mechanism in order to redistribute the surplus when objects are identical (homogeneous objects case). Their mechanism is deterministic, anonymous, and has maximum efficiency over all possible Groves redistribution mechanisms. We will refer to their mechanism as the *worst case optimal* (WCO) mechanism. The WCO Mechanism is a linear redistribution mechanisms. In this paper, we concentrate on designing a linear redistribution mechanism for the heterogeneous objects case.

1.1. Relevant Work

As it is impossible to achieve allocative efficiency, DSIC, and budget balance simultaneously, we have to compromise on one of these properties. Faltings [4] and Guo and Conitzer [9] achieve budget balance by compromising on AE. If we are interested in preserving AE and DSIC, we have to settle for a non-zero surplus or a non-zero deficit of the money (budget imbalance) in the system. To reduce budget imbalance, various rebate functions have been designed by Bailey [1], Cavallo [2], Moulin [12], and Guo and Conitzer [8]. Moulin [12] and Guo and Conitzer [8] designed a Groves redistribution mechanism for assignment of p homogeneous objects among $n > p$ agents with unit demand. Guo and Conitzer [10] designed a redistribution mechanism which is optimal in the expected sense for the homogeneous objects setting. Thus, it will require some distributional information over the type sets of the agents. Gujar and Narahari [7] have designed a non-linear rebate function in the heterogeneous settings. However they only conjectured that the rebate function has a non-zero efficiency and is worst case optimal. To the best of our knowledge, linear rebate functions have not been investigated in the heterogeneous settings.

1.2. Contributions and Outline

In this paper, we investigate the question of existence of a linear rebate function for redistribution of surplus in assignment of heterogeneous objects. Our result shows that in general, when the domain of valuations for each agent is \mathbb{R}_+^p , it is impossible to design a linear rebate function, with non-zero efficiency, for the heterogeneous settings. However, we can relax the assumption of independence of valuations

of different objects to get a linear rebate function with non-zero efficiency. Another way to get around the impossibility theorem is to relax the linearity requirement of a rebate function. In particular, our contributions in this paper can be summarized as follows.

- We first prove the impossibility of existence of a linear rebate function with non-zero efficiency for the heterogeneous settings when the domain of valuations for each agent is \mathbb{R}_+^p and the valuations for the objects are independent.
- When the objects are heterogeneous but the values for the objects of an agent can be derived from one single number, that is, the private information is still single dimensional, we design a Groves redistribution mechanism which is linear, anonymous, deterministic, feasible, individually rational, and efficient. In addition, the mechanism is worst case optimal.
- We show the existence of a non-linear rebate function that has non-zero efficiency. This is different from the rebate function presented in [7] which is only conjectured to have non-zero efficiency.

The paper is organized as follows. In Section 2, we introduce the notation followed in the paper and describe some background work from the literature. In Section 3, we state and prove the impossibility result. We derive an extension of the WCO mechanism for heterogeneous objects but with single dimensional private information in Section 4. The impossibility result does not rule out possibility of non-linear rebate functions with strictly positive efficiency. We show this with a redistribution mechanism, BAILEY, which is Bailey's mechanism [1] applied to the settings under consideration in Section 5. We will conclude the paper in Section 6. We need an ordering of the bids of the agents which we define in Appendix A.

2. Preliminaries and Notation

The notation used is summarized in Table 2. Note that, where the context is clear, we will use t, t_i, r_i, k , and v_i to indicate $t(b), t_i(b), r_i(b), k(b)$, and $v_i(k(b))$ respectively. In this paper, we assume that the payment made by agent i is of the form $t_i(\cdot) - r_i(\cdot)$, where $t_i(\cdot)$ is agent i 's payment in the Clarke pivotal mechanism [3]. We refer to $\sum_i t_i$, as the total Clarke payment or the surplus in the system.

2.1. Optimal Worst Case Redistribution when Objects are Identical

When the objects are identical, every agent i has the same value for each object, call it v_i . Without loss of generality, we will assume, $v_1 \geq v_2 \geq \dots \geq v_n$. In Clarke pivotal mechanism, the first p agents will receive the objects and each of these p agents will pay v_{p+1} . So, the surplus in the

n	Number of agents
N	Set of the agents = $\{1, 2, \dots, n\}$
p	Number of objects
i	Index for an agent, $i = 1, 2, \dots, n$
j	Index for object, $j = 1, 2, \dots, p$
\mathbb{R}_+	Set of positive real numbers
Θ_i	The space of valuations of agent i , $= \mathbb{R}_+^p$
b_i	Bid submitted by agent i , $= (b_{i1}, b_{i2}, \dots, b_{ip}) \in \Theta_i$
b	(b_1, b_2, \dots, b_n) , the bid vector
K	The set of all allocations of p objects to n agents, each getting at most one object
$k(b)$	An allocation, $k(\cdot) \in K$, corresponding to the bid profile b
$k^*(b)$	An allocatively efficient allocation when the bid profile is b
$k_{-i}^*(b)$	An allocatively efficient allocation when the bid profile is b and agent i is excluded from the system
$v_i(k(b))$	Valuation of the allocation k to the agent i , when b is the bid profile
v	$v : K \rightarrow \mathbb{R}$, the valuation function, $v(k(b)) = \sum_{i \in N} v_i(k(b))$
$t_i(b)$	Payment made by agent i in the Clarke pivotal mechanism, when the bid profile is b , $t_i(b) = v_i(k^*(b)) - (v(k^*(b)) - v(k_{-i}^*(b)))$
$t(b)$	The Clarke payment, that is, the total payment received from all the agents, $t(b) = \sum_{i \in N} t_i$
t^{-i}	The Clarke payment received in the absence of the agent i
$r_i(b)$	Rebate to agent i when bid profile is b
e	The efficiency of the mechanism, $= \inf_{b: t \neq 0} \frac{\sum r_i(b)}{t(b)}$

Table 1. Notation

system is pv_{p+1} . For this situation, Moulin [12] and Guo and Conitzer [8] have independently designed a redistribution mechanism.

Guo and Conitzer [8] maximize the worst case fraction of the total surplus which gets redistributed. This mechanism is called the WCO mechanism. Moulin [12] minimizes the ratio of budget imbalance to the value of an optimal allocation, that is the value of an allocatively efficient allocation. The WCO mechanism coincides with Moulin's feasible and individually rational mechanism. Both the above mechanisms work as follows. After receiving bids from the agents, bids are sorted in decreasing order. The first p agents receive the objects. Each agent's Clarke payment is calculated, say t_i . Every agent i pays, $p_i = t_i - r_i$, where, r_i is the rebate function for an agent i . Suppose $y_1 \geq y_2 \geq \dots \geq y_{n-1}$ are the bids of the $(n-1)$ agents excluding the agent i , then the rebate to the agent i is given by,

$$r_i^{WCO} = \sum_{j=p+1}^{n-1} c_j y_j \quad (1)$$

where,

$$c_j = \frac{(-1)^{j+p-1} (n-p) \binom{n-1}{p-1}}{j \binom{n-1}{j} \sum_{k=p}^{n-1} \binom{n-1}{k}} \left\{ \sum_{k=j}^{n-1} \binom{n-1}{k} \right\} \quad (2)$$

for $j = p+1, \dots, n-1$.

The efficiency of this mechanism is e^* , where e^* is given by,

$$e^* = 1 - \frac{\binom{n-1}{p}}{\sum_{k=p}^{n-1} \binom{n-1}{k}}$$

This has been shown to be optimal in the sense that no other mechanism can guarantee greater than e^* fraction redistribution in the worst case.

3. Impossibility of Linear Rebate Function with Non-Zero Efficiency

We have just reviewed the design of a redistribution mechanism for homogeneous objects. We have seen that the WCO mechanism is a linear function of the types of agents. We now explore the general case. In the homogeneous case, the bids are real numbers which can be arranged in decreasing order. The Clarke surplus is a linear function of these ordered bids. For the heterogeneous scenario, this would not be the case. Each bid b_i belongs to \mathbb{R}_+^p ; hence, there is no unique way of defining an order among the bids. Moreover, the Clarke surplus is not a linear function of the received bids in the heterogeneous case. So, we cannot expect any linear/affine rebate function of types to work well at all type profiles. We will prove this formally.

We first generalize a theorem due to Guo and Conitzer [8]. The context in which Guo and Conitzer [8] stated and proved the theorem is in the homogeneous setting. We show that this result holds true in the heterogeneous objects case

also. The symbol \succ denotes the order over the bids of the agents, as defined in the Appendix A.2.

Theorem 3.1: Any deterministic, anonymous rebate function f is DSIC iff,

$$r_i = f(v_1, v_2, \dots, v_{i-1}, v_{i+1}, \dots, v_n) \quad \forall i \in N \quad (3)$$

where, $v_1 \succ v_2 \succ \dots \succ v_n$.

Proof: We provide only a sketch of the proof.

- The “if” part: If r_i takes the form given by equation (3), then the rebate of agent i is independent of his valuation. The allocation rule satisfies allocative efficiency. So, the mechanism is still Groves and hence DSIC. The rebate function defined is deterministic. If two agents have the same bids, then, as per the ordering defined in Appendix, \succ , they will have the same ranking. Suppose agents i and $i + 1$ have the same bids. Thus $v_i \succ v_{i+1}$ and $v_{i+1} \succ v_i$. So, $r_i = f(v_1, v_2, \dots, v_{i-1}, v_{i+1}, \dots, v_n)$ and $r_{i+1} = f(v_1, v_2, \dots, v_i, v_{i+2}, \dots, v_n)$. Since $v_i = v_{i+1}$, $r_i = r_{i+1}$. Thus the rebate function is anonymous.
- The “only if” part: The homogeneous objects case is a special case of the mechanism. When objects are homogeneous, the ordering of the bids \succ matches the \geq ordering on real numbers. If the rebate function is not in the form defined in the theorem, the rebate function would not simultaneously satisfy the DSIC, anonymity, and deterministic properties. This is because the above form of the rebate function is a necessary condition when the objects are identical. Thus we need a rebate function in this form in heterogeneous settings as well. \square

We now state and prove the main result of this paper.

Theorem 3.2: If a redistribution mechanism is feasible and individually rational, then there cannot exist a linear rebate function which satisfies all the following properties:

- DSIC
- deterministic
- anonymous
- non-zero efficiency.

Proof : Assume that there exists a linear function, say f , which satisfies the above properties. Let $v_1 \succ v_2 \succ \dots \succ v_n$. Then according to Theorem 3.1, for each agent i ,

$$\begin{aligned} r_i &= f(v_1, v_2, \dots, v_{i-1}, v_{i+1}, \dots, v_n) \\ &= (c_0, e_p) + (c_1, v_1) + \dots + (c_{n-1}, v_n) \end{aligned}$$

where, $c_i = (c_{i1}, c_{i2}, \dots, c_{ip}) \in \mathbb{R}^p$, $e_p = (1, 1, \dots, 1) \in \mathbb{R}^p$, and (\cdot, \cdot) denotes the inner product of two vectors in \mathbb{R}^p . Now, we will show that the worst case performance of f will be zero. To this end, we will study the structure of f , step by step.

Observation 1: Consider type profile (v_1, v_2, \dots, v_n) where $v_1 = v_2 = \dots = v_n = (0, 0, \dots, 0)$. For this type profile,

the total Clarke surplus is zero and $r_i = (c_0, e_p) \quad \forall i \in N$. Individual rationality implies,

$$(c_0, e_p) \geq 0 \quad (4)$$

Feasibility should imply the total redistributed amount is less than the surplus, that is,

$$\sum_i r_i = n(c_0, e_p) \leq 0 \quad (5)$$

From, (4) and (5), it is easy to see that, $(c_0, e_p) = 0$.

Observation 2: Consider type profile (v_1, v_2, \dots, v_n) where $v_1 = (1, 0, 0, \dots, 0)$ and $v_2 = \dots, v_n = (0, 0, \dots, 0)$. For this type profile, $r_1 = 0$ and if $i \neq 1$, $r_i = c_{i1} \geq 0$ for individual rationality. For this type profile, the Clarke surplus is zero. Thus, for feasibility, $\sum_i r_i = (n-1)c_{11} \leq t = 0$. This implies, $c_{11} = 0$.

In the above profile, by considering $v_1 = (0, 1, 0, \dots, 0)$, we get $c_{12} = 0$. Similarly, one can show $c_{13} = c_{14} = \dots = c_{1p} = 0$.

Observation 3: Continuing like above with, $v_1 = v_2 = \dots = v_i = e_p$, and $v_{i+1} = (1, 0, \dots, 0)$ or $(0, 1, 0, \dots, 0), \dots$ or $(0, \dots, 0, 1)$, we get, $c_{i+1} = (0, 0, \dots, 0) \quad \forall i \leq p-1$. Thus,

$$r_i = \begin{cases} (c_{p+1}, v_{p+2}) + \dots + (c_{n-1}, v_n) & : \text{if } i \leq p+1 \\ (c_{p+1}, v_{p+1}) + \dots + (c_{i-1}, v_{i-1}) \\ \quad + (c_i, v_{i+1}) + \dots + (c_{n-1}, v_n) & : \text{otherwise} \end{cases}$$

We now claim that the efficiency of this mechanism is zero. That is, in the worst case, the fraction of the Clarke surplus that gets redistributed is zero. Suppose we show that there exists a type profile, for which the Clarke surplus is non-zero and the rebate to each agent is zero. Then the theorem is proved. So, it remains to show the existence of such a type profile. Consider the type profile:

$$\begin{aligned} v_1 &= (2p-1, 2p-2, \dots, p) \\ v_2 &= (2p-2, 2p-3, \dots, p-1) \\ &\vdots \\ v_p &= (p, p-1, \dots, 1) \end{aligned} \quad (6)$$

and $v_{p+1} = v_{p+2} \dots = v_n = (0, 0, \dots, 0)$.

Now, with this type profile, agent 1 pays $(p-1)$, agent 2 pays $(p-2), \dots$, agent $(p-1)$ pays 1 and the remaining agents pay 0. Thus, the Clarke payment received is non-zero but it can be seen that $r_i = 0$ for all the agents. \square

The above theorem provides disappointing news. It rules out the possibility of a linear redistribution mechanism for the heterogeneous settings which will have non-zero efficiency. However, there are two ways to get around it.

- 1) The domain of types under which Theorem 3.2 holds is, $\Theta_i = \mathbb{R}_+^p, \quad \forall i \in N$. One idea is to restrict the domain of types. In Section 4, we design a worst

case optimal linear redistribution mechanism when the valuations of agents for the heterogeneous objects have a certain type of relationship.

- 2) Explore the existence of a rebate function which is not a linear and yields a non-zero performance. We explore this in Section 5.

4. A Redistribution Mechanism for Heterogeneous Objects When Valuations have Scaling Based Relationship

Consider a scenario where the objects are not identical but the valuations for the objects are related and can be derived by a single parameter. As a motivating example, consider the website `somefreeads.com` and assume that there are p slots available for advertisements and there are n agents interested in displaying their ads. Naturally, every agent will have a higher preference for a higher slot. Define *click through rate* of a slot as the number of times the ad is clicked, when the ad is displayed in that slot, divided by the number of impressions. Let the click through rates for slots be $\alpha_1 \geq \alpha_2 \geq \alpha_3 \dots \geq \alpha_p$. Assume that each agent has the same value for each click by the user, say v_i . So, the agent's value for the j^{th} slot will be $\alpha_j v_i$. Let us use the phrase *valuations with scaling based relationship* to describe such valuations. We define this more formally below.

Definition 4.1: We say the valuations of the agents have scaling based relationship if there exist positive real numbers $\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_p > 0$ such that, for each agent $i \in N$, the valuation for object j , say θ_{ij} , is of the form $\theta_{ij} = \alpha_j v_i$, where $v_i \in \mathbb{R}_+$ is a private signal observed by agent i . Without loss of generality, we assume, $\alpha_1 \geq \alpha_2 \geq \alpha_3 \dots \geq \alpha_p > 0$. We immediately note that the homogeneous setting is a special case that arises when $\alpha_1 = \alpha_2 = \alpha_3 = \dots = \alpha_p > 0$

For the above setting, we design a Groves mechanism which is almost budget balanced and optimal in the worst case. Our mechanism is similar to that of Guo and Conitzer [8] and our proof uses the same line of arguments.

The following theorem by Guo and Conitzer [8] will be used to design our mechanism.

Theorem 4.1: For given n and real numbers a_1, a_2, \dots, a_n , for any $x_1 \geq x_2 \geq \dots \geq x_n \geq 0$,

$$a_1 x_1 + a_2 x_2 + \dots + a_n x_n \geq 0 \text{ iff } \sum_{i=1}^j a_i \geq 0 \quad \forall j = 1, 2, \dots, n$$

4.1. The Proposed Mechanism

We will use a linear rebate function. (For simplifying equations, we will assume that there are $(n - p)$ virtual objects, with $\alpha_{p+1} = \alpha_{p+2} = \dots = \alpha_n = 0$). We propose the following mechanism:

- The agents submit their bids.
- The bids are sorted in decreasing order.
- The highest bidder will be allotted the first object, the second highest bidder will be allotted the second object, and so on.
- Agent i will pay $t_i - r_i$, where t_i is the Clarke payment and r_i is the rebate.

$$t_i = \sum_{j=i}^p (\alpha_j - \alpha_{j+1}) v_{j+1}$$

- Let agent i 's rebate be,

$$r_i = c_0 + c_1 v_1 + \dots + c_{i-1} v_{i-1} + c_i v_{i+1} + \dots + c_{n-1} v_n$$

The mechanism is required to be individually rational and feasible.

- The mechanism will be individually rational iff $r_i \geq 0 \quad \forall i \in N$. That is, $\forall i \in N$,

$$c_0 + c_1 v_1 + \dots + c_{i-1} v_{i-1} + c_i v_{i+1} + \dots + c_{n-1} v_n \geq 0.$$

- The mechanism will be feasible if the total redistributed payment is less than or equal to the surplus. That is, $\sum_i r_i \leq t$ or $t - \sum_i r_i \geq 0$, where,

$$t = \sum_{j=1}^p j (\alpha_j - \alpha_{j+1}) v_{j+1}.$$

With the above setup, we now derive c_0, c_1, \dots, c_{n-1} that will maximize the fraction of the surplus which is redistributed among the agents.

Step 1: First, we claim that, $c_0 = c_1 = 0$. This can be proved as follows. Consider the type profile, $v_1 = v_2 = \dots = v_n = 0$. For this type profile, individual rationality implies $r_i = c_0 \geq 0$ and $t = 0$. So for feasibility, $\sum_i r_i = n c_0 \leq t = 0$. That is, c_0 should be zero. Similarly, by considering type profile $v_1 = 1, v_2 = \dots = v_n = 0$, we get $c_1 = 0$. □

Step 2: Using $c_0 = c_1 = 0$,

- The feasibility condition can be written as:

$$\sum_{j=2}^{n-1} \left\{ (j-1)(\alpha_{j-1} - \alpha_j) - (j-1)c_{j-1} - (n-j)c_j \right\} v_j - (n-1)c_{n-1} v_n \geq 0 \quad (7)$$

- The individual rationality condition can be written as $c_2 v_2 + \dots + c_{i-1} v_{i-1} + c_i v_{i+1} + \dots + c_{n-1} v_n \geq 0$ (8)

Step 3: When we say our mechanism's efficiency is e , we mean, $\sum_i r_i \geq e t$, that is,

$$\sum_{j=2}^{n-1} \left(-e(j-1)(\alpha_{j-1} - \alpha_j) + (j-1)c_{j-1} + (n-j)c_j \right) v_j + (n-1)c_{n-1} v_n \geq 0 \quad (9)$$

Step 4: Define $\beta_1 = \alpha_1 - \alpha_2$, and for $i = 2, \dots, p$, let $\beta_i = i(\alpha_i - \alpha_{i+1}) + \beta_{i-1}$. Now, inequalities (7), (8), and (9) have to be satisfied for all values of $v_1 \geq v_2 \geq \dots \geq v_n$. Invoking Theorem (4.1), we need to satisfy the following set of inequalities:

$$\begin{aligned}
 \sum_{i=2}^j c_i &\geq 0 \quad \forall j = 2, \dots, n-1 \\
 e\beta_1 &\leq (n-2)c_2 \leq \beta_1 \\
 e\beta_{i-1} &\leq n \sum_{j=2}^{i-1} c_j + (n-i)c_i \leq \beta_{i-1} \quad i = 3, \dots, p \\
 e\beta_p &\leq n \sum_{j=2}^{p-1} c_j + (n-p)c_p \leq \beta_p \quad i = p+1, \dots, n-1 \\
 e\beta_p &\leq n \sum_{j=2}^{n-1} c_j \leq \beta_p
 \end{aligned} \tag{10}$$

Now, the social planner wishes to design a mechanism that maximizes e subject to the above constraints.

Define $x_j = \sum_{i=2}^j c_i$ for $j = 2, \dots, n-1$. This is equivalent to solving the following linear program.

maximize e
s.t.

$$\begin{aligned}
 e\beta_1 &\leq (n-2)x_2 \leq \beta_1 \\
 e\beta_{i-1} &\leq ix_{i-1} + (n-i)x_i \leq \beta_{i-1} \quad i = 3, \dots, p \\
 e\beta_p &\leq ix_{i-1} + (n-i)x_i \leq \beta_p \quad i = p+1, \dots, n-1 \\
 e\beta_p &\leq nx_{n-1} \leq \beta_p \\
 x_i &\geq 0 \quad \forall i = 2, \dots, n-1
 \end{aligned}$$

(11)

So, given n and p , the social planner will have to solve the above optimization problem and determine the optimal values of $e, c_2, c_3, \dots, c_{n-1}$. It would be of interest to derive a closed form solution for the above problem.

The discussion above can be summarized as the following theorem.

Theorem 4.2: When the valuations of the agents have scaling based relationship, for any p and $n > p+1$, the linear redistribution mechanism obtained by solving LP (11) is worst case optimal among all Groves redistribution mechanisms that are feasible, individually rational, deterministic, and anonymous.

Proof: This can be proved following the line of arguments of Guo and Conitzer [8]. □

5. Non-linear Redistribution Mechanisms for the Heterogeneous Setting

We should note that the homogeneous objects case is a special case of the heterogeneous objects case in which each bidder submits the same bid for all objects. Thus, we cannot expect any redistribution mechanism to perform better than the homogeneous objects case. For $n \leq p+1$, the worst case redistribution is zero for the homogeneous case and so will be for the heterogeneous case. So, we assume $n > p+1$. We construct a redistribution scheme by applying the mechanism proposed by Bailey [1] to the heterogeneous settings. We refer to this proposed mechanism on heterogeneous objects

as **BAILEY** redistribution mechanism. It is crucial to note that the non-zero efficiency of the BAILEY mechanism does not trivially follow from that of the mechanism in [1].

5.1. BAILEY Mechanism

First, consider the case when $p = 1$. Let the valuations of the agents for the object be, $v_1 \geq v_2 \geq \dots \geq v_n$. The agent with the highest valuation will receive the object and would pay the second highest bid. Cavallo [2] proposed the rebate function as,

$$\begin{aligned}
 r_1 &= r_2 = \frac{1}{n}v_3 \\
 r_i &= \frac{1}{n}v_2 \quad i > 2
 \end{aligned} \tag{12}$$

Motivated by this scheme, we propose a scheme for the heterogeneous setting. Suppose agent i is excluded from the system. Then let t^{-i} be the Clarke surplus in the system (defined in Table 2). Define,

$$r_i^B = \frac{1}{n}t^{-i} \quad \forall i \in N \tag{13}$$

- As the Clarke surplus is always positive, $r_i^B \geq 0$ for all i . Thus, this scheme satisfies individual rationality.
- $t^{-i} \leq t \quad \forall i$ (revenue monotonicity). So, $\sum_i r_i^B = \sum_i \frac{1}{n}t^{-i} \leq n \frac{1}{n}t = t$. Thus, this scheme is feasible.

We now show that the BAILEY scheme has non-zero efficiency if $n \geq 2p+1$. First we state two lemmas. The proof will be given in Appendix B. These lemma's are useful in designing redistribution mechanisms for the heterogeneous settings as well as in analysis of the mechanisms. Lemma 2 is used to show non-zero efficiency of the BAILEY mechanism. Lemma 1 is used to find an allocatively efficient outcome for the settings under consideration. Also, this lemma 1 is useful in determining the Clarke payments.

Lemma 1: If we sort the bids of all the agents for each object, then

- 1) An optimal allocation, that is an allocatively efficient allocation, will consist of the agents having bids among the p highest bids for each object.
- 2) Consider an optimal allocation k^* . If any of the p agents receiving objects in k^* is dropped, then there always exists an allocation k_{-i}^* that is an optimal allocation (on the remaining $n-1$ agents) which allocates objects to the remaining $(p-1)$ agents. The objects that these $p-1$ agents receive in k_{-i}^* , may not however be the same as the objects they are allocated in k^* .

Lemma 2: There are at most $2p$ agents involved in deciding the Clarke payment.

Note: When the objects are identical, the bids of $(p+1)$ agents are involved in determining the Clarke payment.

Now, we show non-zero efficiency of the BAILEY redistribution scheme.

Proposition 1: The BAILEY redistribution scheme has non-zero efficiency, if $n \geq 2p+1$.

Proof: In Lemma 2, we have shown that there will be at most $2p$ agents involved in determining the Clarke surplus. Thus, given a type profile, there will be $(n - 2p)$ agents, for whom, $t^{-i} = t$ and this implies that at least $\frac{n-2p}{n}t$ will be redistributed. \square

Note: The proof of Proposition 1 indicates that the efficiency of this mechanism is at least $\frac{n-2p}{n}$.

6. Conclusion

We addressed the problem of assigning p heterogeneous objects among $n > p$ competing agents. When the valuations of the agents are independent of each other and their valuations for each object are independent of valuations on the other objects, we proved the impossibility of the existence of a linear redistribution mechanism with non-zero efficiency. Then we explored two approaches to get around this impossibility. In the first approach, we showed that linear rebate functions with non-zero efficiency are possible when the valuations for the objects have scaling based relationship. In the second approach, we showed that rebate functions with non-zero efficiency are possible if linearity is relaxed.

It would be interesting to see if we can characterize the situations under which linear redistribution mechanisms with non-zero efficiency are possible for heterogeneous settings. Another interesting problem to explore is to design redistribution mechanisms that are worst case optimal for heterogeneous settings.

References

- [1] M. J. Bailey, "The demand revealing process: To distribute the surplus," *Public Choice*, vol. 91, no. 2, pp. 107–26, April 1997.
- [2] R. Cavallo, "Optimal decision-making with minimal waste: strategyproof redistribution of VCG payments," in *AAMAS '06: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*. New York, NY, USA: ACM, 2006, pp. 882–889.
- [3] E. Clarke, "Multi-part pricing of public goods," *Public Choice*, vol. 11, pp. 17–23, 1971.
- [4] B. Faltings, "A budget-balanced, incentive-compatible scheme for social choice," in *Agent-Mediated Electronic Commerce, AMEC*. Springer, 2005, pp. 30–43.
- [5] J. R. Green and J. J. Laffont, *Incentives in Public Decision Making*. Amsterdam: North-Holland Publishing Company, 1979.
- [6] T. Groves, "Incentives in teams," *Econometrica*, vol. 41, pp. 617–631, 1973.
- [7] S. Gujar and Y. Narahari, "Redistribution of VCG payments in assignment of heterogeneous objects," in *Proceedings of 4th International Workshop on Internet and Network Economics, WINE 2008*. Springer, 2008, pp. 438–445.

- [8] M. Guo and V. Conitzer, "Worst-case optimal redistribution of VCG payments," in *EC '07: Proceedings of the 8th ACM conference on Electronic Commerce*. New York, NY, USA: ACM, 2007, pp. 30–39.
- [9] —, "Better redistribution with inefficient allocation in multi-unit auctions with unit demand," in *EC '08: Proceedings of the 9th ACM conference on Electronic commerce*. New York, NY, USA: ACM, 2008, pp. 210–219.
- [10] —, "Optimal-in-expectation redistribution mechanisms," in *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 1047–1054.
- [11] J. Laffont and E. Maskin, "A differential approach to expected utility maximizing mechanisms," in *Aggregation and Revelation of Preferences*, J. J. Laffont, Ed., 1979.
- [12] H. Moulin, "Almost budget-balanced VCG mechanisms to assign multiple objects," *Journal of Economic Theory*, 2008, in Press.
- [13] W. Vickrey, "Counterspeculation, auctions, and competitive sealed tenders," *Journal of Finance*, vol. 16, no. 1, pp. 8–37, March 1961.

Appendix A.

Ordering of the Agents Based on Bid Profiles

We will define a ranking among the agents. This ranking is used crucially in proving Theorem 3.1 on rebate function. This theorem is similar to Cavallo's theorem on characterization of DSIC, deterministic, anonymous rebate functions for homogeneous objects. We would not be actually computing the order among the bidders. We will use this order for proving impossibility of the linear rebate function with the desired properties.

A.1. Properties of the Ranking System

When we are defining ranking/ordering among the agents, we expect the following properties to hold true:

- Any permutation of the objects and the corresponding permutation on bid vector, $(b_{i1}, b_{i2}, \dots, b_{ip})$ for each agent i , should not change the ranking. That is, the ranking should be independent of the order in which the agents are expected to bid for this objects.
- Two bidders with the same bid vectors should have the same rank.
- By increasing the bid on any of the objects, the rank of an agent should not decrease.

A.2. Ranking among the Agents

This is a very crucial step. First, find out all feasible allocations of the p objects among the n agents, each

agent receiving at most one object. Sort these allocations, according to the valuation of an allocation. Call this list \mathcal{L} . To find the ranking between i and j , we use the following algorithm.

- 1) $\mathcal{L}_{ij} = \mathcal{L}$
- 2) Delete all the allocations from \mathcal{L}_{ij} which contain both i and j .
- 3) Find out the first allocation in \mathcal{L}_{ij} which contains one of the agents i or j . Say k' .
 - a) Suppose this allocation contains i and has value strictly greater than any of the remaining allocations from \mathcal{L}_{ij} containing j , then we say, $i \succ j$.
 - b) Suppose this allocation contains j and has value strictly greater than any of the remaining allocations from \mathcal{L}_{ij} containing i , then we say, $j \succ i$.
- 4) If the above step is not able to decide the ordering between i and j , let $\mathcal{A} = \{k \in K | v(k) = v(k')\}$. Update $\mathcal{L}_{ij} = \mathcal{L}_{ij} \setminus \mathcal{A}$ and recur to step (2) till EITHER
 - there is no allocation containing the agent i or j
 - OR
 - the ordering between i and j is decided.
- 5) If the above steps do not give either of $i \succ j$ or $j \succ i$, we say, $i \equiv j$ or $i \succcurlyeq j$ as well as $j \succcurlyeq i$

Before we state some properties of this ranking system \succcurlyeq , we will explain it with an example. Let there be two items A and B, and four bidders. That is, $p = 2, n = 4$ and let their bids be: $b_1 = (4, 5), b_2 = (2, 1), b_3 = (1, 4)$, and $b_4 = (1, 0)$.

Now, allocation $(A = 1, B = 3)$ has the highest valuation among all the allocations. So,

$$\begin{aligned} \text{agent 1} &\succ \text{agent 2} \\ \text{agent 1} &\succ \text{agent 4} \\ \text{agent 3} &\succ \text{agent 2} \\ \text{agent 3} &\succ \text{agent 4} \end{aligned} \quad (14)$$

Now, in \mathcal{L}_{13} defined in the procedure above, the allocation $(A = 2, B = 1)$ has strictly higher value than any other allocation in which the agent 3 is present. So,

$$\text{agent 1} \succ \text{agent 3}.$$

Thus,

$$\begin{aligned} \text{agent 1} &\succ \text{agent 3} \succ \text{agent 2} \text{ and} \\ \text{agent 1} &\succ \text{agent 3} \succ \text{agent 4} \end{aligned}$$

In \mathcal{L}_{24} , the allocation $(A = 2, B = 1)$ has strictly higher value than any other allocation in which the agent 4 is present. Thus, the ranking of the agents is,

$$\text{agent 1} \succ \text{agent 3} \succ \text{agent 2} \succ \text{agent 4}$$

We can show that the ranking defined above, satisfies the following properties.

- 1) \succcurlyeq defines a total order on set of bids.
- 2) \succcurlyeq is independent of the order of the objects.

- 3) If two bids are the same, then they are equivalent in this order.
- 4) By increasing a bid, no agent will decrease his rank. If agent $i \succcurlyeq$ agent j , we will also say $v_i \succcurlyeq v_j$.

Appendix B.

B.1. Proof of Lemma 1

Proof:

- Suppose an optimal allocation contains an agent whose bid for his winning object, say j , is not in the top p bids for the j^{th} object. There are other $(p - 1)$ winners in an optimal allocation. So, there exists at least one agent whose bid is in the top p bids for the j^{th} object and does not win any object. Thus, allocating him the j^{th} object, we have an allocation which has higher valuation than the declared optimal allocation.
- Suppose an agent i who receives an object in an optimal allocation is removed from the system. The agent will have at most one bid in the top p bids for each object. So, agents now having bids in the top p bids, will be at the p^{th} position. It can be seen that there will be at most one agent in an optimal allocation who is on the p^{th} position for the object he wins. If there is more than one agent in an optimal allocation on the p^{th} position for the object they win, then we can improve on this allocation. Hence, after removing i , there will be at most one more agent who will be a part of a new optimal allocation. \square

B.2. Proof of Lemma 2

Proof: The argument is as follows:

- 1) Sort the bids of the agents for each object.
- 2) The optimal allocation consists of agents having bids in the p highest bids for each of the objects (Lemma 1).
- 3) For computing the Clarke payment of the agent i , we remove the agent and determine an optimal allocation. And, using his bid, the valuation of optimal allocation with him and without him will determine his payment. This is done for each agent i . As per Lemma 1, if any agent from an optimal allocation is removed from the system, there exists a new optimal allocation which consists of at least $(p - 1)$ agents who received the objects in the original optimal allocation.
- 4) There will be p agents receiving the objects and determining their payments will involve removing one of them at a time, there will be at most p more agents who will influence the payment. Thus, there are at most $2p$ agents involved in determining the Clarke payment. \square

Talking Your Way into Agreement: Belief Merge by Persuasive Communication

Alexandru Baltag
Computing Laboratory
Oxford University
Email: Alexandru.Baltag@comlab.ox.ac.uk

Sonja Smets
Dept. of Artificial Intelligence,
and Dept. of Philosophy
University of Groningen
&
IEG, Oxford University
Email: S.J.L.Smets@rug.nl

I. INTRODUCTION

We investigate the issue of *reaching doxastic agreement* among the agents of a group by “*sharing*” information via successive acts of *sincere, persuasive public communication* within the group.

As usually considered in Social Choice theory, the problem of *preference aggregation* is to find a natural and fair “*merge*” operation (subject to various naturalness or fairness conditions), for aggregating the agents’ preferences into a single group preference. Depending on the stringency of the required fairness conditions, one can obtain either an Impossibility theorem (e.g. Arrows theorem [2]) or a classification of the possible types of reasonable merge operations [1].

In this paper we propose a more “*dynamic*” approach to this issue. Dynamically speaking, “*merging*” preference relations means finding an action or a sequence of actions (a protocol) that, when applied to any arbitrary multi-agent preference model, produces a new model in which all the agents’ preference relations are the same. When the new relations are the result of a specific merge operation, we say that we have “*realized*” this operation via the given (sequence of) action(s). One would like to know what types of merges are realizable by using only specific types of preference-changing actions.

In a doxastic/epistemic setting, the agents preference relations are interpreted as “*doxastic preferences*” or “*doxastic plausibility*” orders. These encode the agents beliefs, but in fact they capture *all* their doxastic-epistemic attitudes: their “*knowl-*

edge” (in the sense of absolutely certain, un-revisable, irrevocable knowledge, i.e. the epistemic concept mostly used in Logic, Computer Science and Economics), their “*strong beliefs*” and “*safe beliefs*” (also known as “*defeasible knowledge*”, i.e. the epistemic concept used mostly by philosophers and researchers in Belief Revision theory), as well as their “*conditional beliefs*” (encoding their “*belief-revision strategy*”, i.e. their contingency plans for belief change). In other words, an agent’s doxastic preference structure capture *all her “information”*: both her “*hard*” (absolutely certain, infallible) information and her “*soft*” (potentially fallible) information. In this context, a preference merge operation corresponds to a way of *combining the agents information into a single “group information”*.

Similarly, preference-changing actions can be interpreted in a doxastic setting as acts of *communication* or *persuasion*. But not every preference-changing action can be understood in this way: there has to be a specific relation between one agent’s (the speaker’s) prior preferences before the action and the whole’s group’s posterior preferences. Actions in which this relation holds will be instances of *sincere and persuasive public communication*.

An announcement of some information P is said to be “*public*” when it is common knowledge that this particular message P is announced and that all the agents are adopting the same attitude towards the (plausibility of the) announcement: they all adopt the same opinion about the reliability of this information. Depending on the specific common

attitude, there are three main possibilities that have been discussed in the literature: 1) the information P is *certainly* true: it is *common knowledge* that the message is necessarily truthful; (2) the announcement is *strongly believed* by all agents to be true: it is *common knowledge that everybody strongly believes* that the speaker tells the truth; (3) the announcement is (*simply*) *believed*: it is *common knowledge that everybody believes (in the simple, “weak” sense)* that the speaker tells the truth. These three alternatives correspond to three forms of “learning” a public announcement, forms discussed in [12], [14] in a Dynamic Epistemic Logic context: “update” $^1 !P$, “radical upgrade” $\uparrow P$ and “conservative upgrade” $\uparrow P$. Under various names, they have been previously proposed in the literature on Belief Revision, e.g. by Rott [23] and Boutilier [10], and in the literature on dynamic semantics for natural language by Veltman [28]. The first operation (update) models a “*truthful public announcements*” of “*hard*” information; the other two are models of “*soft*” public announcements.

“Sincerity” of a communication act can be defined as sharing of information that *was already “accepted” by the speaker* (before the act). The meaning of “acceptance” depends on the form of communication: as we’ll see, for updates with “hard” information, acceptance means “knowledge” (in the irrevocable sense), while for upgrades with “soft” information, acceptance just means some type of “belief” or “strong belief” (depending on whether the upgrade is “conservative” or “radical”). But, as a general concept, prior acceptance requires that the speaker’s own doxastic structure should *not* be changed by her sincere communication.

“Persuasiveness” requires that the communicated information *becomes commonly “accepted” by all the agents* (in the same sense of “acceptance” that the speaker has adopted): this means that, after the act, everybody *commonly* exhibits *the same doxastic attitude as the speaker* (knowledge, belief or strong belief) towards the communicated information. So,

after a persuasive communication, all agents reach a *partial agreement*, namely with respect to the specific information that has been communicated.

In a cooperative setting, the *goal* of “sharing” doxastic information is *reaching “agreement”* with respect to *all* the (relevant) issues. Indeed, the natural stopping point of iterated sharing is when *nothing is left for further sharing or persuading*; i.e. *complete* agreement. Any further sincere persuasive communication is *redundant* at that stage: it can no longer change any agent’s doxastic structure. This happens exactly when all the agents’ relevant doxastic attitudes towards all issues are *exactly the same*. (Which attitude is relevant depends again of the type of communication: “knowledge” for updates, “belief” for conservative upgrades, “strong belief” for radical upgrades). This means that the agents’ (relevant) accessibility relations (i.e. respectively, the knowledge relations, the belief structure or the strong belief structure) *became identical*: we say that these structures *have “merged” into one*.

So we arrive in a natural way at the main issue addressed in this paper: *the “dynamic merge” of doxastic structures by sincere persuasive public communication*. In particular, we investigate the realizability of merge operations via (1) updates, (2) radical upgrades and (3) conservative upgrades. We show that, in the first case, only the epistemic structures (given by the “hard” knowledge relations) can be merged; and moreover, the only form of realizable merge is in this case the so-called “parallel merge” [1], given by the intersection of all preference relations. Epistemically, this corresponds to the familiar concept of “distributed knowledge”. The realizability result is constructive, it comes with a specific announcement-based protocol for realizing this merge. This is essentially the algorithm in van Benthem’s paper “One is a Lonely Number” [11]: the agents announce “all they know”, in no particular order. In the second case (radical upgrade), the “defeasible knowledge” structures are merged, but in fact this implies that all the other doxastic attitudes become the same: *the agents’ whole “doxastic preference” structures are merged*. The natural analogue of the above-mentioned protocol for radical upgrades realizes now a different type of merge (“priority merge”, itself a natural epistemic modification of the other basic type of

¹Note that in Belief Revision, the term “belief update” is used for a totally different operation (the Katzuno-Mendelzon update[21]), while what we call “update” is known as “conditioning”. We choose to follow here the terminology used in Dynamic Epistemic Logic, but we want to warn the reader against any possible confusions with the KM update.

merge considered in [1], the “lexicographic merge”). Finally, in the case of conservative upgrades, only the (simple) belief structures (given by the doxastic relations) can be merged. Moreover, priority merge is realizable via the natural analogue of the same protocol above for conservative upgrades.

This *surface similarity* between the three cases is pleasing, but in fact it hides *deeper dissimilarities*. As we mentioned, the realizable merge is *unique* in the first case. This is *not* true in the other cases: a whole class of merge operations can be realized by radical or conservative upgrades. Moreover, in the first case, the order in which the announcements is irrelevant, while in the other cases *the order matters*: if the upgrades are performed in a different order than the one prescribed in the protocol, then different merge operations may be realized! Finally, in the first case, the merge may be realized by allowing *only one announcement by each agent* (of “all she knows”). But this is *not true* in the other cases: the agents may have to make many soft announcements, including announcing facts that may already be entailed by their previous announcements!

Some of the questions we address in this paper came to our attention after hearing a presentation by J. van Benthem on “The Social Choice Behind Belief Revision” at the workshop “Dynamic Logic Montreal” in 2007 [13]. Van Benthem’s view was that belief dynamics in itself can be captured as a form of preference merge (between the prior doxastic preferences and the on-going doxastic preferences about the new information). One can see that our approach here is actually *the dual* of the perspective adopted in [13]: implementing preference merge dynamically by successive belief revisions, instead of understanding belief revision in terms of preference merge.

In the next section we introduce the necessary background on different notions of knowledge, belief and other doxastic attitudes. The main focus will be on the semantics, which is given via preference models. In section III, we introduce the main concepts of belief dynamics, following the work in [3], [4], [5], [6], [12] on joint upgrades, as models for “sincere, persuasive public announcements”. In section IV we present three natural merge operations: parallel merge, lexicographic merge and

(relative) priority merge. In section V we present the protocols for dynamic realizations of parallel merge and priority merge, giving counterexamples that point out the differences between them. We end with a short note and an open question in our Conclusions section.

II. PLAUSIBILITY STRUCTURES AND DOXASTIC ATTITUDES

In this section, we review some basic notions and results from [3]. We use finite “plausibility” frames, in the sense of our papers [3], [4], [5], [6], [7], [8]. These kind of semantic structures are the *natural multi-agent generalizations of structures that are standard*, in one form or another, in Belief Revision: Halpern’s “preferential models” [20], Spohn’s ordinal-ranked models [24], Board’s “belief-revision structures” [16], Grove’s “sphere” models [19]. Unlike the settings in [7], [8], we restrict here to the *finite* case, for reasons of simplicity.

For a given set \mathcal{A} of labels called “agents”, a (*finite, multi-agent*) *plausibility frame* is just a finite, multi-agent Kripke frame $(S, R_a)_{a \in \mathcal{A}}$ in which the accessibility relations $R_a \subseteq S \times S$ are usually denoted by \leq_a , are called “plausibility orders” or “doxastic preference” relations, and are assumed to be *locally connected preorders*. Here, a “locally connected preorder” $\leq \subseteq S \times S$ is a reflexive and transitive relation such that: if $s \leq t$ and $s \leq w$ then either $t \leq w$ or $w \leq t$; and if $t \leq s$ and $w \leq s$ then either $t \leq w$ or $w \leq t$. See [3] for a justification and motivation for these conditions.² We use the notation $s \sim_a t$ for the *comparability relation* with respect to \leq_a (i.e. $s \sim_a t$ iff either $s \leq_a t$ or $t \leq_a s$), $s <_a t$ for the corresponding *strict order relation* (i.e. $s <_a t$ iff $s \leq_a t$ but $t \not\leq_a s$), and $s \cong_a t$ for the corresponding *indifference relation* (i.e. $s \cong_a t$ iff both $s \leq_a t$ and $t \leq_a s$). When using the R_a notation for the preference relations \leq_a , we also use the notations $R_a^<$, R_a^\sim and R_a^\cong to denote the corresponding strict order, comparability and indifference relations $<_a$, \sim_a and \cong_a .

In a plausibility frame, the comparability relations \sim_a are *equivalence relations*, hence they induce *partitions*. We denote by $s(a) := \{t \in S : s \sim_a t\}$ the

²In the infinite case, one has to add a well-foundedness condition, obtaining “locally well-preordered” relations.

\sim_a -partition cell of s , comprising all a 's epistemic alternatives for s . Finally, we use \rightarrow_a to denote the “best alternative” or “most preferred” relation \rightarrow_a , given by: $s \rightarrow_a t$ iff $t \in s(a)$ and $t \geq_a t'$ for all $t' \in s(a)$.

Plausibility Models A (finite, multi-agent, pointed) plausibility model is a structure $\mathbf{S} = (S, \leq_a, \|\cdot\|, s_0)_{a \in \mathcal{A}}$, consisting of a plausibility frame $(S, \leq_a)_{a \in \mathcal{A}}$ together with a valuation map $\|\cdot\| : \Phi \rightarrow \mathcal{P}(S)$, mapping every element p of some given set Φ of “atomic sentences” into a set of states $\|p\| \subseteq S$, and together with a designated state $s_0 \in S$, called the “actual state”.

(Common) Knowledge and (Conditional) Belief Given a plausibility model \mathbf{S} , sets $P, Q \subseteq S$ of states, an agent $a \in \mathcal{A}$ and some group $G \subseteq \mathcal{A}$, we define: $best_a P = Max_{\leq_a} P := \{s \in P : t \leq_a s \text{ for all } t \in P\}$, $K_a P := \{s \in S : s(a) \subseteq P\}$, $B_a P := \{s \in S : best_a s(a) \subseteq P\}$, $B_a^Q P := \{s \in S : best_a (s(a) \cap Q) \subseteq P\}$, $Ek_G P := \bigcap_{a \in G} K_a P$, $Eb_G P := \bigcap_{a \in G} B_a P$, $Ck_G P := \bigcap_{n \in \mathbb{N}} Ek_G^n P$ (where $Ek_G^0 P := P$ and $Ek_G^{n+1} := Ek_G(Ek_G^n P)$), $Eb P := Eb_{\mathcal{A}} P$, and $Ck P := Ck_{\mathcal{A}} P$.

Interpretation. The elements of S represent the “possible worlds”, or possible states of a system: possible descriptions of the real world. The correct description of the real world is given by the “actual state” s_0 . The atomic sentences $p \in \Phi$ represent “ontic” (non-doxastic) facts, that might hold or not in a given state. The valuation tells us which facts hold at which worlds. For each agent a , the equivalence relation \sim_a represents the agent a 's epistemic indistinguishability relation, inducing a 's information partition; $s(a)$ is the state s 's information cell with respect to a 's partition: if s were the real state, then agent a would consider all the states $t \in s(a)$ as “epistemically possible”. $K_a P$ is the proposition “agent a knows P ”: observe that this is indeed the same as Aumann's partition-based definition of knowledge. The plausibility relation \leq_a is agent a 's “doxastic preference” relation: her plausibility order between her “epistemically possible” states. So we read $s \leq_a t$ as “agent a considers t at least as plausible as s (though the two are epistemically indistinguishable)”. This is meant to capture the agent's (conditional) beliefs about the state of the system. Note that $s \leq_a t$ implies $s \sim_a t$,

so that the agent only compares the plausibility of states that are epistemically indistinguishable: so we are not concerned here with counterfactual beliefs (going against the agent's knowledge), but only with conditional beliefs (if given new evidence that must be compatible with prior knowledge). So $B_a^Q P$ is read “agent a believes P conditional on Q ” and means that, if a would receive some further (certain) information Q (to be added to what she already knows) then she would believe that P was the case. So conditional beliefs B_a^Q give descriptions of the agent's plan (or commitments) about what would she believe (about the current state) if she would learn some new information Q . To quote J. van Benthem in [12], conditional beliefs are “static pre-encodings” of the agent's potential belief changes in the face of new information. The above definition says that $B_a^Q P$ holds iff P holds in all the “best” (i.e. the most plausible) Q -states (that are consistent with a 's knowledge). In particular, a simple (non-conditional) belief $B_a P$ holds iff P holds in all the best states that are epistemically possible for a .

Kripke Modalities For any binary accessibility relation $R \subseteq S \times S$ and set $P \subseteq S$, the corresponding Kripke modality is given by:

$$[R]P := \{s \in S : \forall t (sRt \Rightarrow t \in P)\}$$

We think of sets $P \subseteq S$ as propositions and write $s \models P$ instead of $s \in P$.

It is easy to see that belief is the Kripke modality $B_a = [\rightarrow_a]$ for the “best alternative” relation \rightarrow_a defined above. Similarly, knowledge is the Kripke modality for the epistemic relation $K_a = [\sim_a]$.

Safe belief as “defeasible knowledge” The Kripke modality for the plausibility relation $\square_a := [\leq_a]$ was called “safe belief” in [3], and “the preference modality” in [15]. It was also considered by Stalnaker in [25], as a formalization of Lehrer's notion of “defeasible knowledge”. According to this so-called *defeasibility theory of knowledge*, a belief counts as “knowledge” if it is *stable under belief revision with any true information*. Indeed, the safe belief modality has the property that it is *conditionally believed under any true condition*:

$$s \models \square_a Q \text{ iff: } s \models B_a^P Q \text{ for all } P \text{ such that } s \models P.$$

For this reason, we'll refer to \square using either of the terms “safe belief” and “defeasible knowledge”.

In contrast, the knowledge concept captured by the K modality can be called “irrevocable knowledge”, since it is a belief that is stable under revision with *any* information (including false ones):

$$s \models K_a Q \text{ iff: } s \models B_a^P Q \text{ for all } P.$$

There are other differences: irrevocable knowledge K satisfies the axioms of the modal system $S5$, so it is fully introspective; in contrast, defeasible knowledge \square is only positively introspective, but not necessarily positively introspective. (In fact, the complete logic of \square is the modal logic $S4.3$.) *An agent’s belief can be safe without him necessarily “knowing” this* (in the “strong” sense of the irrevocable knowledge K): “safety” (similarly to “truth”) is an *external* property of the agent’s beliefs, that can be ascertained only by comparing his belief-revision system with reality. Indeed, *the only way for an agent to know a belief to be safe is to actually know it to be truthful*. This is captured by the valid identity: $K_a \square_a P = K_a P$. In other words: *knowing that something is safe to believe is the same as just knowing it to be true*. In fact, *all beliefs held by an agent “appear safe” to him*: in order to believe them, he has to believe that they are safe. This is expressed by the valid identity: $B_a \square_a P = B_a P$, saying that: *believing that something is safe to believe is the same as just believing it*³. Contrast this with the situation concerning “knowledge”: in our logic (as in most standard doxastic-epistemic logics), we have the identity: $B_a K_a P = K_a P$. So *believing that something is known is the same as knowing it!*

The difference between K and \square and their different properties, expressed by the above identities, are enough to solve the so-called “Paradox of the Perfect Believer” in [18], [29], [27], [22], [30], [17]: when we say that somebody “only believes that she knows something (without really knowing it)”, we’re using the word “knowledge” in a different sense than the fully introspective K modality. A natural reading is to interpret it as the defeasible knowledge \square , in which case “believing

that you know” is the same as “believing”, by the identity $B_a \square_a P = B_a P$.

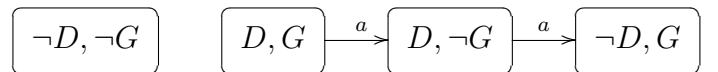
“Strong Belief” Another important doxastic attitude, called *strong belief*, is given by:

$$Sb_a P = \{s \in P : s(a) \cap P \neq \emptyset \text{ and } w >_a t \text{ for all } t \in s(a) \cap P \text{ and all } w \in s(a) \setminus P\}.$$

So P is strong belief at a state s iff P is *epistemically possible* and moreover *all epistemically possible P -states at s are more plausible than all epistemically possible non- P states*. This notion was called “strong belief” by Battigalli and Siniscalchi [9], while Stalnaker [26] calls it “robust belief”. It is easy to see that we have the following equivalence: $S \models Sb_a P$ iff $S \models B_a P$ and $S \models B_a^Q P$ for every Q such that $S \models \neg K_a(Q \rightarrow \neg P)$. In other words: *something is strong belief iff it is believed and if this belief can only be defeated by evidence (truthful or not) that is known to contradict it*. An example is the “presumption of innocence” in a trial: requiring the members of the jury to hold the accused as “innocent until proven guilty” means asking them to start the trial with a “strong belief” in innocence.

Example 1: Consider the situation of Professor Albert Winestein. Albert feels that he is a genius. He knows that there are only two possible explanations for this feeling: either he is a genius or he’s drunk. He doesn’t feel drunk, so he believes that he is a sober genius. However, *if he realized that he’s drunk*, he’d think that his genius feeling was just the effect of the drink; i.e. *after learning he is drunk* he’d come to *believe that he was just a drunk non-genius*. In reality though, Albert is *both drunk and a genius*.

We can represent Albert’s information and (conditional) by the following plausibility relation:

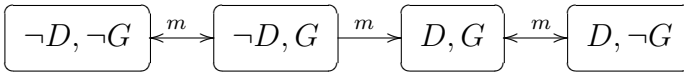


Here, as in all other drawings, we use *labeled arrows for plausibility relations \leq_a* (not for the “best alternative” relations \rightarrow_a !), going from less plausible to more plausible worlds, but *we skip loops and composed arrows* (since \leq_a are reflexive and transitive). The real world is (D, G) . Albert considers $(D, \neg G)$ as being more plausible than (D, G) , and $(\neg D, G)$ as more plausible than $(D, \neg G)$. Albert can distinguish all these worlds

³The proof is an easy semantic exercise, which can be rendered in English as: saying that “the best worlds have the property that all the worlds at least as good as them are P -worlds” is equivalent to simply saying that “the best worlds are P -worlds”.

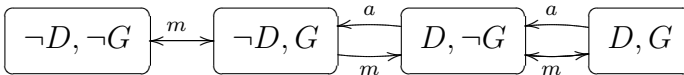
from $(\neg D, \neg G)$, since (in the real world) he knows (K_a) that either D or G holds.

Consider another agent, Professor Mary Curry. She is pretty sure that Albert is drunk: she can see this with her very own eyes. But Marry is *completely indifferent with respect to Albert's genius*: so she considers the possibility of genius and the one of non-genius as equally plausible. However, having a philosophical mind, Mary is aware of the possibility that the testimony of her eyes may in principle be wrong: it is in principle possible that Albert is not drunk, despite the presence of the usual symptoms. Marry's beliefs are captured by her plausibility order:



We can see from the drawing that Mary strongly believes D , and in fact her belief is safe: so she “knows” that Albert is drunk, in the sense of defeasible knowledge (although she doesn't know it, in the sense of K). But she is completely indifferent with respect to G : hence she considers the possibility of G and $\neg G$ as equally plausible.

To put together the agents' plausibility orders, we need to be told what do they know about each other. Suppose all their opinions as described above (i.e. all their conditional beliefs) are *common knowledge*: essentially, this means their doxastic preferences are common knowledge. We thus obtain the following multi-agent plausibility model:

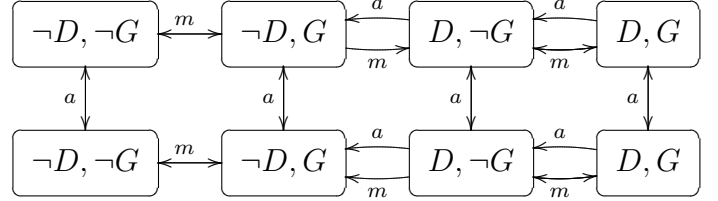


At the real world (D, G) , one can check that $B_a G$ is true. Further, Albert does not know G , hence $(D, G) \models \neg K_a G \wedge \neg \Box_a G$ while $(D, G) \models K_a(D \vee G)$. Moreover, he doesn't “know” G in the defeasible sense either: his belief in G is *not safe*, since $B_a^D \neg G$ holds in the real world: so if Albert would learn (correctly) that he was drunk, he'd lose his (true) belief in being a genius.

Example 2 Let us now relax our assumptions about the agents' mutual knowledge: suppose that only Albert's opinions are common knowledge; in addition, suppose that it is common knowledge that Mary has no opinion on Albert's genius (so she considers genius and non-genius as equi-plausible), but that

she has a strong opinion about his drunkenness: she can see him, so judging by his behavior she either strongly believes he's drunk or she strongly believes he's not drunk. However, her actual opinion about this is unknown to Albert, who thus considers both opinions as equally plausible.

The resulting model is:



The real world is represented by *the upper* (D, G) state. One can check that, in the real world, Mary still strongly believes Albert he's drunk; but he does not know this: Mary's plausibility relation between D and $\neg D$ is unknown to Albert. However, he knows that either she strongly believes D or she strongly believes $\neg D$.

We can go on and modify the example further, by allowing that Albert's plausibility is not commonly known either etc. But, for simplicity of drawing, we stop here: when less common knowledge is assumed, more worlds are possible, and hence the drawings get more and more complex.

G -Bisimulation For a group $G \subseteq \mathcal{A}$ of agents, we say the pointed models $\mathbf{S} = (S, \geq_a, \|\|, s_0)_{a \in \mathcal{A}}$ and $\mathbf{S}' = (S', \geq'_a, \|\|', s'_0)_{a \in \mathcal{A}}$ are *G -bisimilar*, and write $\mathbf{S} \simeq_G \mathbf{S}'$, if the pointed Kripke models $(S, \geq_a, \|\|, s_0)_{a \in G}$ and $(S', \geq'_a, \|\|', s'_0)_{a \in G}$ (having as accessibility relations only the G -labeled relations) are bisimilar in the usual sense from Modal Logic [?]. When $G = \mathcal{A}$, we simply write $\mathbf{S} \simeq \mathbf{S}'$, and say \mathbf{S} and \mathbf{S}' are *bisimilar*. Bisimilar models *differ only formally*: they encode precisely *the same* doxastic-epistemic information, and they satisfy the same modal sentences.

III. BELIEF DYNAMICS: SINCERE, PERSUASIVE PUBLIC COMMUNICATION

We move on now to *belief dynamics*: what happens when some proposition P is *publicly announced*? According to Dynamic Epistemic Logic, this induces, not only a revision of beliefs, but a *change of model*: a “revision” of the whole relational structure, changing the agents' plausibility

orders. However, the specific change depends on *the agents' attitudes to the plausibility* of the announcement: *how certain is the new information?* Three main possibilities have been discussed in the literature: (1) the announcement P is *certainly* true: it is *common knowledge* that the speaker tells the truth; (2) the announcement is *strongly believed* to be true by everybody: it is *common knowledge that everybody strongly believes* that the speaker tells the truth; (3) the announcement is (*simply*) *believed*: it is *common knowledge that everybody believes (in the simple, "weak" sense)* that the speaker tells the truth. These three alternatives correspond to three forms of "joint learning", forms discussed in [12], [14] in a Dynamic Epistemic Logic context: "update" $^4 !P$, "radical upgrade" $\uparrow P$ and "conservative upgrade" $\uparrow P$. Under various names, the single-agent versions of these doxastic transformers have been previously proposed by e.g. Rott [23], Boutilier [10] and Veltman [28].

We will use "joint upgrades" as a general term for all these three model transformers, and denote them in general by $\dagger P$, where $\dagger \in \{!, \uparrow, \uparrow\}$. Formally, each of our joint upgrades is a (possibly partial) function taking as inputs pointed models $\mathbf{S} = (S, \leq_a, \|\cdot\|, s_0)$ and returning new ("upgraded") pointed models $\dagger P(\mathbf{S}) = (S', \leq'_a, \|\cdot\|', s'_0)$, with $S' \subseteq S$. Since upgrades are purely doxastic, they *won't affect the real world or the "ontic facts" of each world*: i.e. they all satisfy $s'_0 = s_0$ and $\|p\|' = \|p\| \cap S'$, for atomic p . So, in order to completely describe a given upgrade, we only have to specify (a) *its possible inputs* \mathbf{S} , (b) *the new set of states* S' ; (c) *the new relations* \leq'_a .

(1) Learning Certain information: Joint "Update". The update $!P$ is an operation on pointed models which is *executable* (on a pointed model \mathbf{S}) *iff* P is true (at \mathbf{S}) and which *deletes all the non- P -worlds from the pointed model, leaving everything else the same*. Formally, an update $!P$ is an upgrade such that: (a) it takes as inputs only pointed models \mathbf{S} , such that $\mathbf{S} \models P$; (b) the new set of states

is $S' = \{s \in S : s \models P\}$; (c) $s \leq'_a t$ iff $s \leq_a t$ and $s, t \in S'$.

(2) Learning from a Strongly Trusted Source: (Joint) "Radical" Upgrade. The "radical upgrade" (or "lexicographic upgrade") $\uparrow P$, as an operation on pointed plausibility models, can be described as "promoting" *all the P -worlds within each information cell so that they become "better" (more plausible) than all $\neg P$ -worlds in the same information cell, while keeping everything else the same*: the valuation, the actual world and the relative ordering between worlds within either of the two zones (P and $\neg P$) stay the same. Formally, a radical upgrade $\uparrow P$ is (a) a *total* upgrade (taking as input *any* model \mathbf{S}), such that (b) $S' = S$, and (c): $s \leq'_a t$ iff either $s \notin P_{\mathbf{S}}$ and $t \in s(a) \cap P_{\mathbf{S}}$, or $s \leq_a t$.

(3) "Barely believing" what you hear: (Joint) "Conservative" Upgrade. The so-called "conservative upgrade" $\uparrow P$ (called "minimal conditional revision" by Boutilier [10]) performs in a sense the minimal possible revision of a model that is forced by believing the new information P . As an operation on pointed models, it can be described as "promoting" *only the "best" (most plausible) P -worlds, so that they become the most plausible in their information cell, while keeping everything else the same*. Formally, $\uparrow P$ is (a) a *total* upgrade, such that (b) $S' = S$, and (c): $s \leq'_a t$ iff either $t \in \text{best}_a(s(a) \cap P_{\mathbf{S}})$ or $s \leq_a t$.

Redundancy, Informativity and Sincerity A joint upgrade $\dagger P$ is *redundant on a model \mathbf{S} with respect to a group of agents $G \subseteq \mathcal{A}$* if the upgraded model is G -bisimilar to the original one: $\dagger P(\mathbf{S}) \simeq_G \mathbf{S}$. This means that, as far as the group G is concerned, $\dagger P$ doesn't change anything: all the group G 's doxastic attitudes *stay the same* after the upgrade. An upgrade $\dagger P$ is *informative (on \mathbf{S}) to group G* if it is not redundant with respect to G . An upgrade $\dagger P$ is *redundant with respect to an agent a* if it is redundant with respect to the singleton $\{a\}$.

Redundancy is especially important if we want to capture the "sincerity" of an announcement made by a speaker. Intuitively, an announcement is "sincere" when it agrees with the speaker's prior epistemic state: *accepting the announcement doesn't change the speaker's own state*.

Definition: A (public) announcement $\dagger \varphi$ made

⁴Note that in Belief Revision, the term "belief update" is used for a totally different operation (the Katzuno-Mendelzon update[21]), while what we call "update" is known as "conditioning". We choose to follow here the terminology used in Dynamic Epistemic Logic, but we want to warn the reader against any possible confusions with the KM update.

by an agent a is said to be *sincere* if it leaves unchanged agent a 's own plausibility structure; i.e. it's *non-informative* to agent a .

Proposition 1

- 1) In a pointed model \mathbf{S} , $!P$ is *redundant* with respect to a group G iff P is *common knowledge* in \mathbf{S} among the group G ; i.e.: $\mathbf{S} \simeq_G !P(\mathbf{S})$ iff $\mathbf{S} \models Ck_G P$. Special case: an announcement $!P$ made by an agent a is *sincere* iff a knows P , i.e. if $K_a P$ holds in the original model (before the announcement).
- 2) $\uparrow P$ is *redundant* with respect to a group G iff it is *common knowledge in the group G that P is strongly believed* (by all G -agents); i.e. $\mathbf{S} \simeq_G \uparrow P(\mathbf{S})$ iff $\mathbf{S} \models Ck_G(ESb_G)$. Special case: an announcement $\uparrow P$ made by an agent a is *sincere* iff a strongly believes P (before the announcement).
- 3) $\uparrow P$ is *redundant* with respect to a group G iff it is *common knowledge in the group G that P is believed* (by all G -agents); i.e. $\mathbf{S} \simeq_G \uparrow P(\mathbf{S})$ iff $\mathbf{S} \models Ck_G(Eb_G P)$. Special case: an announcement $\uparrow P$ made by an agent s is *sincere* iff s believes P (before the announcement).

Invariance under communication: For a given upgrade type $\dagger \in \{!, \uparrow, \uparrow\}$, we say that a pointed model \mathbf{S} is *invariant under \dagger -communication within group G* iff, for all propositions P , any sincere announcement of the form $\dagger P$ made by any agent in G is redundant in \mathbf{S} .

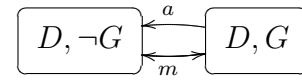
Proposition 2

- 1) A pointed model \mathbf{S} is *invariant under $!$ -communication within G* iff all (irrevocable) knowledge is common knowledge within G , i.e. for all propositions P and all agents $a, b \in \mathcal{A}$, $K_a P$ holds in \mathbf{S} iff $K_b P$ holds in \mathbf{S} ; equivalently: iff all G -agents' epistemic relations coincide: $\sim_a = \sim_b$ for all $a, b \in G$.
- 2) A pointed model \mathbf{S} is *invariant under \uparrow -communication within G* iff all "defeasible knowledge" is common defeasible knowledge within G , i.e. for all propositions P and all agents $a, b \in \mathcal{A}$, $\Box_a P$ holds in \mathbf{S} iff $\Box_b P$ holds in \mathbf{S} ; equivalently: iff all strong beliefs (conditional beliefs) are common strong beliefs (common conditional beliefs); equiva-

lently: iff all G -agents' plausibility relations coincide: $\leq_a = \leq_b$ for all $a, b \in G$.

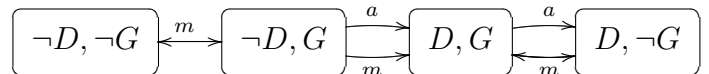
- 3) A pointed model \mathbf{S} is *invariant under \uparrow -communication within G* iff all (simple) beliefs are common beliefs within G , i.e. for all propositions P and all agents $a, b \in \mathcal{A}$, $B_a P$ holds in \mathbf{S} iff $B_b P$ holds in \mathbf{S} ; equivalently: iff all G -agents' "best alternative" relations coincide: $\rightarrow_a = \rightarrow_b$ for all $a, b \in G$.

Example 3 Suppose that in the situation in Example 1 above, a *trusted, infallible source* publicly announces that Albert is drunk: this is "hard", incontrovertible information, corresponding to a *joint update* $!D$. The updated model is



After the update, Albert starts to wrongly believe that $\neg G$ is the case! This is an example of *true but un-safe belief*: it can be lost after acquiring (new) true information.

Example 4 Consider again the situation in example 3, but instead of Albert receiving the information from an infallible source, he receives the information from Mary. Mary announces publicly (to Albert) that D is the case and we assume that Mary's announcement is both *sincere* and *persuasive*: she tells what she thinks and she convinces Albert. Since Mary is a fallible agent (and not an infallible source), this announcement is *soft*: in principle, she could be wrong, or she could lie, or she could simply guess and be right only by chance. So we cannot interpret Mary's announcement as a "hard" update $!D$, since such an announcement wouldn't be sincere: the update $!D$ would automatically change Mary's order (making her irrevocably know D , when she didn't know it before!). But we can model it as a "soft" announcement $\uparrow D$; i.e. after hearing it, all agents upgrade with D : they start to prefer any D -world to any $\neg D$ -world. The upgraded model is



Note that *Mary's order* is left unchanged, so the announcement was indeed *sincere*.

Example 5 What if instead Mary announces that she "knows" that Albert is drunk? If we take this in

the sense of irrevocable knowledge K , then such an announcement would *not* be sincere: indeed, in the original situation of Example 1, $K_m D$ was false. However, she *did* “know” it in the sense of *defeasible knowledge* $\Box_m D$: she correctly believed D , and this belief was safe. This “knowledge” was fallible, and she was aware of this: she didn’t believe that she knows irrevocably ($\neg B_m K_m D$), but she believed that she “knows” defeasibly ($B_m \Box_m D$). Hence, she is sincere if she announces that she “knows” in this sense. Assuming that Albert is also aware of the fallibility of her knowledge, but that he still highly trusts her to be right, we can interpret this as a sincere and persuasive announcement of the form $\uparrow(\Box D)$. Its effect is the same as in Example 4: the upgraded model is the same.

Counterexample 6 Note that *simply announcing that she believes D , or even that she strongly believes D , won’t do: this will not be persuasive*, since it will not change Albert’s beliefs about the facts of the matter (D or $\neg D$), although it may change his beliefs about her beliefs. *Being informed of another’s beliefs is not enough to convince you of their truth.* Indeed, Mary’s beliefs are already common knowledge in the initial model of Example 1: so an upgrade $\uparrow(B_m D)$ would be superfluous!

Persuasiveness So what is needed for persuasive communication is that the speaker (Mary) “*converts*” the others to her own beliefs. For this, she should *not* simply announce that she believes them. Instated, she can either announce that something is the case (when in fact she *just strongly believes* that it is the case), or else announce that she *defeasibly “knows”* it (when she *only believes that she “knows”* it, and in fact this implies that she strongly believes that she “knows”).

IV. MERGE OPERATIONS

A *merge operation*, or “aggregation procedure”, is an operator taking any sequence $\{R_i\}_{1 \leq i \leq n}$ of preference relations into a “group preference” relation $\odot_i R_i = R_1 \odot R_2 \odot \dots \odot R_n$. In [1] the authors give a *general classification* of types of preference merge, in a very general context, subject to some minimal “*fairness*” and *rationality conditions*. They show that all the merge operations satisfying these conditions can be represented as compositions of

only two basic merge operators: “parallel merge” and “lexicographic merge”.

Parallel Merge The merge operation we consider first can be thought of as the *most “democratic” form of aggregation*: everybody has a veto, so that group preferences are unanimous preferences. Following [1], we call it *parallel merge*. It simply takes the merged relation to be the *intersection* $\odot_a R_{a \in G} := \bigcap_{a \in G} R_a$ of all the preference relations of the agents in a given group $G \subseteq \mathcal{A}$.⁵

Parallel merge is particularly well suited for aggregating the agents’ “hard information” (irrevocable knowledge) K , i.e. for merging the epistemic relations $\{\sim_a\}_{a \in G}$. Since if we consider absolutely certain and fully introspective knowledge, there is no danger of introducing an inconsistency. The agents can pool their information in a *completely symmetric manner, accepting the other’s bits without reservations*. In fact, parallel merge of the agents’ irrevocable knowledge gives us the standard concept of “*distributed knowledge*” DK :

$$DK_G P = \left[\bigcap_{a \in G} R_a \right] P.$$

Lexicographic Merge When the group is hierarchically structured according to some total order (on agents), called a “*priority order*”, then the agents with higher priority are thought of as having a higher “epistemic expertise” than the agents with lower priority. For a group $G = \{a, b\}$ of two agents, in which a has higher priority, we can think of a as the “expert” (or the professor) and of b as the “layman” (or the student). In this context, the natural doxastic merge operation is the so-called *lexicographic merge*. For two agents a, b , the “lexicographic merge” $R_{a/b}$ gives priority to agent a ’s strong (i.e. strict) preferences over b ’s: first, the *strict preference order of a is adopted by the group*; and when a is indifferent between two options, then *b ’s preference is adopted*; finally, a -incomparability gives group incomparability. Formally:

$$R_{a/b} := R_a^< \cup (R_a^= \cap R_b) = R_a^< \cup (R_a \cap R_b) = R_a \cap (R_a^< \cup R_b).$$

⁵From a purely formal perspective, parallel merge resembles the so-called “non-prioritized belief revision” known from the work of S. H. Hansson, H. Rott, H. van Ditmarsch. But note that “merge” is *not* “revision”!

The lexicographic merge is *particularly suited for aggregating “soft information”* (strong beliefs, safe beliefs, conditional beliefs) *in the absence of any hard information*: since soft information is not fully reliable (because of lack of negative introspection for \square , and because of potential falsity for belief, conditional belief and strong belief), some “screening” must be applied to some agents’ information (and so some hierarchy must be enforced), in order to ensure consistency of the merge.

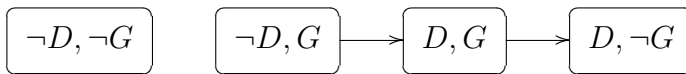
(Relative) Priority Merge Note that, *in lexicographic merge, the first agent’s priority is “absolute”*. But *in the presence of “hard” information*, the lexicographic merge of soft information must be modified, by first pooling together all the hard information and then using it to restrict the lexicographic merge of soft information. This leads us to a “more democratic” combination of Parallel Merge and Lexicographic Merge, called “(relative) priority merge” $R_{a \otimes b}$:

$$R_{a \otimes b} := (R_a^< \cap R_b^{\sim}) \cup (R_a^{\cong} \cap R_b) = R_a \cap R_b^{\sim} \cap (R_a^< \cup R_b).$$

In a Relative Priority Merge, both agents have a “veto” with respect to group incomparability. Here the group can only compare options that *both* agents can compare; and whenever the group can compare two options, everything goes on as in the lexicographic merge. Agent a ’s order gets priority, while b ’s order is adopted only when a is indifferent.

Since our plausibility structures they encode both the “hard” and the “soft” information possessed by the agent, it seems that Priority Merge is best suited for aggregating the agents’ plausibility relations.

Example 7: If in Example 1, we give priority to Mary, the relative priority merge $R_{m \otimes a}$ of Mary’s and Albert’s original plausibility orders amounts to:



If instead we give priority to Albert, we simply obtain Albert’s order as our “merge”:

$$R_{a \otimes m} = R_a.$$

It is important to note that in *both* cases of Example 7, some of the resulting *joint beliefs are wrong*: when giving priority to Mary, both agents end up

believing $\neg G$; while if we give priority to Albert, they both end up believing $\neg D$. In fact, *no type of hierarchic belief merge is a warranty of veracity*.

V. “REALIZING” MERGE DYNAMICALLY

Intuitively, the *purpose of sharing* hard knowledge, defeasible knowledge or beliefs is to achieve a state in which *there is nothing else to share*, i.e. one in which any further sharing is redundant: all hard knowledge, or defeasible knowledge, or beliefs, are already shared in common. For sharing via a specific type of public communication $\dagger \in \{!, \uparrow, \uparrow\}$, this happens precisely when the model-changing process induced by \dagger -type sharing *reaches a fixed point of \dagger -communication*: a model that is invariant under that particular type of announcements.

For every specific type of public communication $\dagger \in \{!, \uparrow, \uparrow\}$, agent a ’s “*relevant structure*” in a model \mathbf{S} is given by: a ’s *epistemic relation* $\overset{a}{\sim} \subseteq S \times S$ in the case of *updates* $!$; a ’s *plausibility relation* \leq_a in the case of *radical upgrade* \uparrow ; and a ’s *doxastic “best alternative” relations* \rightarrow_a in the case of *conservative upgrade*.

A (finite) \dagger -*upgrade sequence* is a finite sequence $\dagger \vec{P} = (\dagger P^1, \dots, \dagger P^n)$ of upgrades $\dagger P^i$ of the given type $\dagger \in \{!, \uparrow, \uparrow\}$. Any \dagger -upgrade sequence induces a (partial) function, mapping every pointed model \mathbf{S} into a finite sequence $\dagger \vec{P}(\mathbf{S}) = (\mathbf{S}^i)_i$ of *pointed models*, defined inductively by: $\mathbf{S}^0 := \mathbf{S}$; and $\mathbf{S}^{i+1} := \dagger P^i(\mathbf{S}^i)$, if this is defined (and undefined otherwise). A \dagger -upgrade sequence $\dagger \vec{P}$ is a \dagger -*communication sequence within group* G if all its upgrades are sincere for at least one G -agent at the moment of speaking: i.e. for every $i \leq n$ there exists $a_i \in G$ such that $\dagger P^i$ is sincere for a_i on \mathbf{S}^i .

A \dagger -communication sequence $\dagger \vec{P}$ within a group G is *exhaustive* on a model \mathbf{S} if the last model \mathbf{S}^n of the induced sequence $\dagger \vec{P}(\mathbf{S})$ is invariant under (sincere) \dagger -communication; equivalently, iff it is *maximal*: it cannot be extended to any longer \dagger -communication sequence. By Proposition 2, the last model \mathbf{S}^n generated by an exhaustive \dagger -communication sequence is one in which *all the G -agents’ “relevant structures”* R_a^n *coincide*.

An exhaustive \dagger -communication sequence within G *realizes a given preference merge operation* \otimes on a given pointed model \mathbf{S} if, for any agent $b \in G$, the relevant structures R_b^n in the last

generated model is the \otimes -merge of the initial relevant structures $\{R_a^0\}_{a \in G}$: i.e. $R_b^n = \otimes_{a \in G} R_a^0$, for all $b \in G$. A merge operation \otimes is *realizable* by \dagger -communication (within a group G) if there exists some exhaustive \dagger -communication sequence (within G) that realizes \otimes . The merge operation is said to be *constructively realizable* by \dagger -communication if *there exists a protocol* such that every \dagger -communication sequence that complies with the protocol is exhaustive and realizes \otimes .

For each of the above types of public communication ($!$, \uparrow , \uparrow), we can ask which merge operations are realizable, or constructively realizable. The answer depends on the constraints (*transitivity, connectedness* etc.) satisfied by the agents' relevant structures (epistemic, doxastic or plausibility relations).

Proposition 3 *Parallel merge is the only merge operation that is realizable by updates* (i.e. by $!$ -communication). Moreover, *parallel merge is constructively realizable by updates*. The protocol is as follows: *in no particular order, the agents have to publicly announce "all that they know"* (in the sense of *irrevocable knowledge* K). More precisely, for each set of states $P \subseteq S$ such that P is *known to a given agent* a , a public announcement $!P$ is made.

This essentially is the protocol in van Benthem's paper "One is a Lonely Number" [11]. Formally, the protocol consists of n steps, each step being a sequence of announcements by the same agent: first, one of the agents, say a , announces all he knows. This is the sequence of announcements:

$$\sigma_a := \prod \{!P : P \subseteq S \text{ such that } s \models K_a P\}$$

(where \prod is sequential composition of actions). Then, another agent b performs a similar step (announcing all she knows after the first step), etc.

Important Observations: (1) *The order in which the agents make the announcements doesn't actually matter. They may even "interrupt" each other: any exhaustive $!$ -communication sequence produces the same result.* (2) The protocol can be simplified by restricting it only to *knowledge announcements*, i.e. of the form $!(K_a P)$ (for each P such $K_a P$ holds): instead of announcing all they know, the agents announce *that they know* all that they know. (3) The protocol can be simplified by *allowing each agent*

to make only one announcement: instead of successively announcing everything he knows, he can just announce the conjunction $!(\bigwedge \{P : S \models K_a P\})$ of all the things he knows.

Proposition 4 *For every given priority order (a_1, \dots, a_n) on agents, the corresponding priority merge (of plausibility relations) is constructively realizable by radical upgrades* (i.e. by \uparrow -communication), *but is not the only such realizable operation*. The protocol is a natural modification of the previous one: *following the priority order, the agents have to publicly announce "all that they strongly believe"*. More precisely, for each set $P \subseteq S$ such that P is *strongly believed by the given agent* a , a joint radical upgrade $\uparrow P$ is performed.

Formally, the protocol consists again of n steps, each step being a sequence of announcements by the same agent: first, the first agent according to the priority order, say a , announces all that he strongly believes. This is the sequence of radical upgrades:

$$\rho_a := \prod \{\uparrow P : P \subseteq S \text{ such that } S \models S b_a P\}.$$

Then, the next agent in the hierarchy, say b , performs a similar step (announcing all she strongly believes *after* the first step), etc.

Important Observations: (1) *Now, the order of the announcements matters*: the agents have to respect the priority order. Moreover, *no interruptions are allowed*: agents with lower priority can speak *only after the agents with higher priority finished* announcing all their strong beliefs. Any interruptions may lead to the realization of *complete different merge operations* (see the Counterexample below)! (2) This protocol can also be simplified by restricting it only to *"defeasible knowledge" announcements*, i.e. announcements of the form $!(\Box_a P)$. But recall that, unlike irrevocable knowledge, defeasible is not negatively introspective: so the agents don't know for sure what things they "know" and what not, and hence the best they can do is to announce all the things they *believe they "know"*. But, since believing to (indefeasibly) "know" is the same as believing, they have to *announce that they "know"* P , for each proposition P which they believe. So the simplified protocol replaces e.g. the first step by the following sequence of radical upgrades

$$\rho'_a := \prod \{\uparrow (\Box_a P) : P \subseteq S \text{ such that } S \models B_a P\}.$$

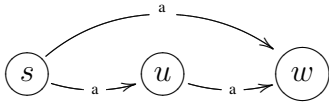
(3) Unlike the case of upgrades and parallel merge, in general the above protocol *actually requires multiple announcements by the same agents, including announcing facts that may already be entailed by their previous announcements!* A sequence of radical upgrades is in general *not* equivalent to a radical upgrade, so there is no way to compress the sequences ρ_a or ρ'_a into a single upgrade!

Example 8 Recall the initial order of Marry and Albert in Example 1. Consider the protocol:

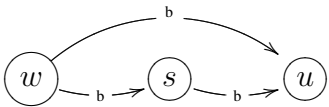
$$\uparrow \Box_m D; \uparrow K_a(D \vee G); \uparrow \Box_a \neg G$$

The first is a sincere announcement by Mary, the rest are sincere announcements by Albert. The second announcement, though not in “defeasible knowledge” form (as required by the simplified protocol in observation 2 above), is equivalent to one in this form, because of the identity: $K_a P = \Box_a K_a P$. This communication sequence yields the model presented in Example 7, as the result of the priority merge $R_{m \otimes a}$ of the two plausibility orders.

Counterexample 9 To show the *non-uniqueness* of priority merge among \uparrow -realizable merge operations and the *order-dependency* of the above protocol, note first that the priority merge of the ordering



with the ordering

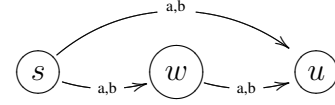


is equal to either of the two orders (depending on which agent has priority). But consider now the following public dialogue

$$\uparrow \Box_b u \cdot \uparrow \Box_a (u \vee w)$$

This first is a *sincere announcement* by b , the second is *sincere announcement* by a . This is an exhaustive \uparrow -communication sequence, but note that the strict priority order required by the above protocol is *not* respected here: the first speaker b is “interrupted” by the second speaker a *before* she finished announcing all his strong beliefs. (Indeed, $s \vee u$ is also a strong belief of agent b , though one that is entailed by

the first announcement; nevertheless, b should have first announced this second strong belief before a would have been allowed to speak!) And, indeed, the resulting model, though a fixed point of \uparrow -communication (since all the plausibility relations come to coincide), *realizes a different merge operation than either of the two priority merges:*



The Power of Agendas This order-dependence illustrates a phenomenon well-known in Social Choice Theory: the *important role of the person who “sets the agenda”*: the “Judge” who assigns *priorities to witnesses’ stands*; the chairman or moderator who determines the *order of the speakers* in a meeting, as well as the *the issues* to be discussed and the *relative priority of each issue*.

Proposition 5 *For every given priority order (a_1, \dots, a_n) on agents, the corresponding priority merge of doxastic “best alternatives” relations $\{\rightarrow_a\}_a$ is constructively realizable by conservative upgrades (i.e. by \uparrow -communication). The protocol is the natural modification of the previous one: following the priority order, the agents have to publicly announce “all that they (simply) believe”. More precisely, for each set of states $P \subseteq S$ such that P is believed by the given agent a , a joint conservative upgrade $\uparrow P$ is performed.*

Similar observations as the ones following Proposition 4 apply to the case of doxastic upgrades: priority merge is *not* the only realizable merge operation; the order of announcements *does* matter; in general, the protocol *may* require *multiple announcements by the same agents*.

VI. CONCLUSION

In this paper, we focused on *dynamically realizing two specific merge operations by public communication*. But, as we saw, depending on the “agenda”, soft announcements *can realize a whole plethora of merge operations*. Nevertheless, *not everything goes*: the requirements imposed on the plausibility relations generally pose restrictions to which kinds of merge are realizable. This raises an important open question: *characterize the class of merge operations realizable by radical (or conservative) upgrades*.

REFERENCES

- [1] H. Andreka, M. Ryan and P-Y. Schobbens “Operators and Laws for Combining Preference Relations”, *Journal of Logic and Computation*, 12(1), 13–53, 2002.
- [2] K.J. Arrow, “A Difficulty in the Concept of Social Welfare”, *Journal of Political Economy*, 58(4), 328-346, 1950.
- [3] A. Baltag and S. Smets, “Conditional doxastic models: a qualitative approach to dynamic belief revision”, *Electronic Notes in Theoretical Computer Science*, 165, 5–21, 2006.
- [4] A. Baltag and S. Smets, “The Logic of Conditional Doxastic Actions: A Theory of dynamic multi-agent belief revision”, in S. Artemov and R. Parikh (eds.), *Proceedings of the Workshop on Rationality and Knowledge*, 13–30, ESSLLI 2006.
- [5] A. Baltag and S. Smets, “Dynamic Belief Revision over Multi-Agent Plausibility Models”, in G. Bonanno, W. van der Hoek, M. Wooldridge (eds.), *Proceedings of the 7th Conference on Logic and the Foundations of Game and Decision (LOFT 2006)*, 11–24, University of Liverpool, 2006.
- [6] A. Baltag and S. Smets, *Probabilistic Dynamic Belief Revision*, in J. van Benthem and S. Ju and F. Veltman (eds.), *Proceedings of LORI'07*, College Publications London, 21–39, 2007.
- [7] A. Baltag and S. Smets, “A Qualitative Theory of Dynamic Interactive Belief Revision”, in G. Bonanno, W. van der Hoek, M. Wooldridge (eds.), *Logic and the Foundations of Game and Decision Theory, Texts in Logic and Games*, 3, 9–58, Amsterdam University Press, 2008.
- [8] A. Baltag and S. Smets, “The Logic of Conditional Doxastic Actions”, in R. van Rooij and K. Apt (eds.), *New Perspectives on Games and Interaction, Texts in Logic and Games*, 4, 9–31, Amsterdam University Press, 2008.
- [9] P. Battigalli and M. Siniscalchi, “Strong Belief and Forward Induction Reasoning”, *Journal of Economic Theory*, 105, 356–391, 2002.
- [10] C. Boutilier, “Iterated Revision and Minimal Change of Conditional Beliefs”, *JPL*, 25(3), 262–305, 1996.
- [11] J.F.A.K. van Benthem, “One is a lonely number”. In P. Koepke Z. Chatzidakis and W. Pohlers, (eds.) *Logic Colloquium 2002*, 96-129, ASL and A.K. Peters, Wellesley MA, 2006.
- [12] J.F.A.K. van Benthem, “Dynamic logic of belief revision”, *JANCL*, 17 (2), 129-155, 2007.
- [13] J.F.A.K. van Benthem, “Priority Product Update as Social Choice” (Expanded version), Unpublished Manuscript, November 2007
- [14] J.F.A.K. van Benthem, *Logical Dynamics of Information and Interaction*, Manuscript, To appear, 2009.
- [15] J.F.A.K. van Benthem and F. Liu, “Dynamic logic of preference upgrade”, *Journal of Applied Non-Classical Logics*, University of Amsterdam, 17 (2), 157–182, 2007.
- [16] O. Board, “Dynamic interactive epistemology”, *Games and Economic Behaviour*, 49, 49–80, 2002.
- [17] N. Friedmann and J.Y. Halpern, “Conditional logics of belief revision”, *Proc. of 12th National Conference in Artificial Intelligence*, AAAI Press, Menlo Park, CA, 915–921, 1994.
- [18] P. Gochet and P. Gribomont, “Epistemic Logic”, D.M. Gabbay and J. Woods (eds.), *Handbook of the History of Logic*, Elsevier, 7, 99–195, 2006.
- [19] A. Grove, “Two modellings for theory change”, *Journal of Philosophical Logic*, v17, 157–170, 1988.
- [20] J.Y. Halpern, *Reasoning about Uncertainty*, MIT Press, Cambridge MA, 2003.
- [21] H. Katsuno and A. Mendelzon, “On the difference between updating a knowledge base and revising it”, *Cambridge Tracts in Theoretical Computer Science*, 183–203, 1992.
- [22] J.-J.Ch. Meyer and W. van der Hoek, *Epistemic Logic for AI and Computer Science*, Cambridge Tracts in Theoretical Computer Science, 41, Cambridge University Press, Cambridge, 1995.
- [23] H. Rott, “Conditionals and theory change: revisions, expansions, and additions” in *Synthese*, 81(1), 91-113, 1989.
- [24] W. Spohn, “Ordinal conditional functions: a dynamic theory of epistemic states”, in W.L. Harper and B. Skyrms (eds.), *Causation in Decision, Belief Change, and Statistics*, vol. II, 105–134, 1988.
- [25] R. Stalnaker, “On Logics of Knowledge and Belief”, *Philosophical Studies*, vol. 128, 169–199, 2006.
- [26] R. Stalnaker, “Knowledge, Belief and Counterfactual Reasoning in Games”, *Economics and Philosophy*, vol. 12, 133–163, 1996.
- [27] W. van der Hoek, “Systems for knowledge and beliefs”, *Journal of Logic and Computation*, 3, nr. 2, 173–195, 1993.
- [28] F. Veltman, “Defaults in Update Semantics”, *Journal of Philosophical Logic*, 25, 221–261, 1996.
- [29] F.P.J.M. Voorbraak, *As Far as I Know*, Utrecht University, Utrecht, NL, *Questiones Informatiae* volume VII, 1993.
- [30] T. Williamson, “Some philosophical aspects of reasoning about knowledge”, *Proceedings of TARK'01*, J. van Benthem (ed.), 97–97, Morgan Kaufmann Publishers, San Francisco, 2001.

Comparing strengths of beliefs explicitly

Dick de Jongh

Institute for Logic, Language and Computation
Universiteit van Amsterdam
Email: D.H.J.deJongh@uva.nl

Sujata Ghosh

Institute of Artificial Intelligence
Rijksuniversiteit Groningen
Email: Sujata.Ghosh@rug.nl

Abstract—Inspired by a similar use in provability logic, formulas $p \succ_B q$ and $p \succ_{=B} q$ are introduced in the existing logical framework for discussing beliefs to express that the strength of belief in p is greater than (or equal to) that in q . This explicit mention of the comparison in the logical language aids in defining several other concepts in a uniform way, viz. older and rather clear concepts like the operators for universality (which possibilities ought to be considered), together with newer notions like plausibility (in the sense of ‘more plausible than not’) and disbelief. Moreover, it assists in studying the properties of the concept of greater strength of belief itself. A heavy part is played in our investigations by the relationship between the standard plausibility ordering of the worlds and the strength of belief ordering. If we try to define the strength of belief ordering in terms of the world plausibility ordering we get some undesirable consequences, so we have decided to keep the relation between the two orderings as light as possible to construct a system that allows for widely different interpretations. Finally, after a brief discussion on the multi-agent setting, we move on to talk about the dynamics - the change of ordering under the influence of hard and soft information.

Index Terms—doxastic logic, belief, disbelief, plausibility

I. INTRODUCTION

Being subject to doubts and dilemmas while making decisions is like second nature to the human mind. The difference in the strengths of beliefs of an agent regarding the occurrence of different events may clear doubts of this kind. In betting on games, people make their choices for putting their money on different teams, based on their strengths of beliefs about which team will win. Similarly, when voting, one’s preference for the candidates is again based on the strength of beliefs about one candidate’s ability to perform compared to the others. Thus, this notion is inherently present in various fields of research like decision theory, game theory and others.

Before proceeding further, let us first consider the following real life situation where comparison of strength of beliefs plays a key role in decision-making for recruitments.

Alice often has applications for jobs in her departmental store. The first time Burt and Cora apply. Alice believes both can do the job, but her belief in Cora being able to do it is stronger than that Burt will be able to do it. She chooses Cora.

The second time Deirdre and Egon apply. She believes that Egon can do the job whereas she is ambiguous about Deirdre: she neither has the belief that Deirdre can do it, nor that she cannot. She chooses Egon.

The third time Fiona and Gregory apply. About both she is ambiguous, but her strength of belief in Gregory being able to do it is stronger than that in Fiona. She chooses Gregory, maybe she has to help him along a little.

The fourth time the applicants are Harold and Irma. She believes neither can do the job. She decides to take neither and hold another round of applications.

All these situations regarding the belief states of Alice can be aptly described, if we talk not only about her beliefs but also compare the strength of her beliefs in the applicants. One can argue that these situations can be described by the very well-studied notion of *preference*, but the essence of describing the mental states of Alice will be lost then. This paper addresses the notion of comparison of the strength of beliefs of an agent directly. A great volume of literature and extensive philosophical debates are available on reasoning about knowledge and belief of agents. This paper adds a new notion to this line of work, viz. comparing the strengths of beliefs, and very pertinently, doing this in a qualitative manner. The ordering introduced here operates on formulas.

The introduction of explicit notions of ordering for comparing strengths of beliefs in the logical language has various applications. It aids in defining several other concepts in a uniform way, viz. older and rather clear concepts like the operators for universality, together with newer notions like plausibility and disbelief. Moreover it assists in studying the properties of the concept of greater strength of belief itself. In the semantics, the question - which worlds are going to be a part of the model, gets in our approach a clearer formal and intuitive understanding. It also becomes more evident that the universality operator cannot be identified with the knowledge operator even if they both share the $S5$ -properties. Above all it has its advantages in an explicit study of the properties of the orderings themselves, semantically and axiomatically. All these investigations can be carried over to a dynamic setting. A pleasant fact is that we can fit the system easily into the framework of dynamic epistemic logic ([1], see also [2]) as explained in section III.

As mentioned in a brief interlude later, the explicit belief ordering also aids in providing an additional feather to the already existing close relationship between beliefs and preferences which is thoroughly discussed in [3].

Before entering into the actual study, let us first discuss the previous work on provability logics which inspired this idea of explicit belief ordering. In Provability Logic, an extensive overview of which can be found in [4], $\Box\varphi$ is interpreted as $\exists x Proof_{\mathbf{PA}}(x, \varphi)$, which means that there is a proof x in the axiomatization \mathbf{PA} (Peano Arithmetic) for the statement φ . With \mathbf{PA} understood, we also write it in the form $\neg \exists x Proof(x, \varphi)$. In this framework, to handle Rosser's form of incompleteness, one introduces witness comparisons of proofs with formulas like $\Box\varphi \preceq \Box\psi := \exists x (Proof(x, \varphi) \wedge \forall y < x \neg Proof(y, \psi))$, which is interpreted as that φ has a proof that is at least as small as any (possible) proof of ψ . Similarly, $\Box\varphi \prec \Box\psi := \exists x (Proof(x, \varphi) \wedge \forall y \leq x \neg Proof(y, \psi))$ means that φ has a proof that is smaller than any (possible) proof of ψ . Logics for these witness comparison formulas were successfully introduced by [5], and the completeness proof for this logic was simplified in [6].

Motivated by the ideas above, formulas $\varphi \succ_B \psi$ and $\varphi \succeq_B \psi$ are introduced in the existing logical framework for discussing beliefs to express that the strength of belief in φ is greater than (or equal to) that in ψ . We should note here that in the Rosser framework proofs of φ and ψ are compared only if one of these proofs really exists, whereas strengths of beliefs are also discussed when neither φ nor ψ are really believed, which makes them less concrete, and therefore we express their comparison as $\varphi \succ_B \psi$, rather than $B\varphi \succ B\psi$. As mentioned earlier, these formulas can be used to express notions like 'disbelief' (the inclination to believe in $\neg\varphi$ is greater than the inclination to believe in φ), and its dual 'more plausible than not', which can be represented by $\neg\varphi \succ_B \varphi$ and $\varphi \succ_B \neg\varphi$, respectively.

Let us now mention some related works in this area. In [7], [8], orderings of formulas are considered but their interpretations are probabilistic in nature. A binary sentential operator is introduced in the language with the intended interpretation 'at least as probable as'. While [8] takes the explicit ordering operator in a simple language consisting of the truth-functional connectives only, [7] discusses this issue in a modal setting. Related as well are Lewis's proposal for a plausibility ordering of formulas [9] and Spohn's work on giving plausibility ordering of possible worlds in terms of ordinal functions [10]. More recently, we find some similar work in the economics literature [11]. Also, the notion of epistemic entrenchment [12] gives a syntactic ordering of formulas, which is studied in connection with belief revision. The ordering influences the abandoning and retaining of formulas when a belief contraction or revision takes place. In our setting the dynamics operates rather differently, as the reader will see.

With this background, we now provide a brief summary regarding the structure of this paper. Explicit belief-

ordering over formulas is introduced in section 2. Several possible interpretations of the belief-ordered formulas, viz. *plausibility* and *disbelief* are discussed, together with the inter-relationship of these ordering formulas and *safe belief*, and also, *preference*. Complete axiomatizations of the new belief logic with explicit ordering (*KD45-O*), with safe belief added (*KD45-OS*), plausibility logic (*P-logic*), logic of belief and plausibility (*BP-logic*) and logic of belief and disbelief (*BD-logic*) are provided. The section ends with a short discussion on the multi-agent setting. Section III brings in dynamics to the whole framework and discusses the influence of hard information as well as soft information over these ordering formulas, and provides complete axiomatizations to the dynamic logics under consideration. The conclusions are drawn in section IV.

II. COMPARING STRENGTH OF BELIEFS EXPLICITLY

Modal logic is a useful tool to study knowledge and belief of human agents, which has been a main issue of concern to philosophers as well as computer scientists. Von Wright's work [13] is generally accepted as initiating this line of research, which was further extended by [14]. Subsequently a huge research area has been developed, trying to provide answers to various philosophical issues as well as aiding into the development of several areas of computer science, like distributed systems, security protocols, database theory and others.

Possible-world semantics [15] has been used to model knowledge as well as belief. An extensive discussion together with all pre-requisite definitions can be found in [16]. In this work we are only concerned with *beliefs* of agents, comparison of their strengths as well as some related notions like *universality*, *safe beliefs*, *plausibility*, *disbelief* and others. Various debates and discussions are still going strong among the philosophers regarding the axioms that characterize belief - for this paper we will stick to the *KD45*-model of belief.

In the following, we talk about Kripke structures as well as the plausibility models [2], [17] as and when needed while talking about beliefs. The readers should note that plausibility models are more general in nature in the sense that one can always build up a *KD45* Kripke structure from them as described in [17].

With this brief overview, we now move on to introduce explicit ordering of beliefs in the logical language, which is the essential new feature of this paper. This explicit mention of such comparison of beliefs provides an informative and uniform way to discuss certain relevant issues like disbeliefs, plausibility and others.

To introduce this comparison of strengths of beliefs explicitly in the logical language, we add new relation symbols to the existing modal language of belief to form the language

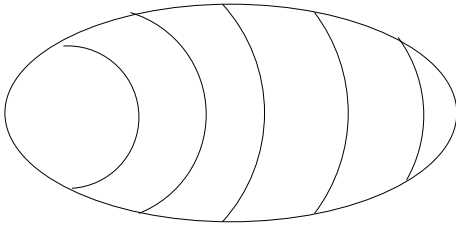


Fig. 1. Plausibility ordering

of *Belief logic with explicit ordering* ($D - O$), whose language is defined as follows:

Definition II.1 Given a countable set of atomic propositions Φ , formulas φ are defined inductively:

$$\varphi := \perp \mid p \mid \neg\varphi \mid \varphi \vee \psi \mid B\varphi \mid \varphi \succ_B \psi \mid$$

where $p \in \Phi$.

The intuitive reading of the formula $B\varphi$ is “ φ is believed”, and that of $\varphi \succ_B \psi$ is “belief in φ is at least as strong as belief in ψ ”. We introduce the notations $\varphi \succ_B \psi$ for $(\varphi \succ_B \psi) \wedge \neg(\psi \succ_B \varphi)$ and $\varphi \equiv_B \psi$ for $(\varphi \succ_B \psi) \wedge (\psi \succ_B \varphi)$. Intuitively, they can be read as “belief in φ is stronger than that in ψ ” and “belief in φ and ψ are of same strength”, respectively. We now move on to define a model for this logic.

Definition II.2 A $D - O$ model is defined to be a structure $\mathcal{M} = (S, \leq, \geq_B, V)$, where S is a non-empty finite set of states, V is a valuation assigning truth values to atomic propositions in states, \leq is a quasi-linear¹ order relation (a plausibility ordering) over S , and \geq_B is a quasi-linear order relation over $\mathcal{P}(S)$, satisfying the conditions

- 1) If $X \subseteq Y$, then $Y \geq_B X$
- 2) If \mathcal{B} is the set of all \leq -minimal worlds (the set of most-plausible worlds, called the center), then $\mathcal{B} \subseteq X$ and $\mathcal{B} \not\subseteq Y$ imply $X >_B Y$, where $X >_B Y$ iff $X \geq_B Y$ and not $(Y \geq_B X)$.
- 3) If X is non-empty, then $X >_B \emptyset$.

The first condition says that larger sets of worlds are more plausible, the second one that the sets containing the center are more plausible than those not containing it, and the third one that non-empty sets are more plausible than the empty set. Truth on the center suffices to make an assertion to be believed. Note that all the models are considered to be finite. This assumption ensures the existence of minimal worlds in terms of the plausibility ordering of the model. The truth definition for formulas φ in a $D - O$ model \mathcal{M} is as usual with the following clauses for the belief and ordering modalities.

$$\mathcal{M}, s \models B\varphi \text{ iff } \mathcal{M}, t \models \varphi \text{ for all } \leq\text{-minimal worlds } t.$$

¹A binary relation \leq on a non-empty set S is said to be quasi-linear if it is reflexive, transitive and linear, i.e. a total pre-order.

$$\mathcal{M}, s \models \varphi \succ_B \psi \text{ iff } \{t \mid \mathcal{M}, t \models \varphi\} \geq_B \{t \mid \mathcal{M}, t \models \psi\}.$$

We consider \succ_B to be a global notion, if $\varphi \succ_B \psi$ is true anywhere in the model, it is true everywhere. So, it is either true or false throughout the whole model; \succ_B is a global notion like B . Of course, being global in the model is strongly connected with introspection. From the definition of \succ_B , it follows that,

$$\mathcal{M}, s \models \varphi \succ_B \psi \text{ iff } \{t \mid \mathcal{M}, t \models \varphi\} >_B \{t \mid \mathcal{M}, t \models \psi\}.$$

Thus, \succ_B is also a global notion. We will now show that the universal modality U can also be expressed in $D - O$. The modality $E\varphi$ (the abbreviated form of $\neg U\neg\varphi$) can be defined as $\varphi \succ_B \perp$, and hence $U\varphi$ itself as $\perp \succ_B \neg\varphi$. To clarify matters we should mention here - $U\varphi$ expresses that φ is true in all possible worlds in the model, whereas $E\varphi$ stands for existence of a possible world in the model where φ is true. The formula $\varphi \succ_B \perp$ which defines $E\varphi$ expresses clearly which worlds should be considered in the model: those worlds of which the existence is expressed by a positive strength of belief, those possibilities which the agent does not want to exclude. Evidently, we have,

$$\mathcal{M}, s \models U\varphi \text{ iff } \mathcal{M}, t \models \varphi \text{ for all worlds } t.$$

Thus U is definable in the language above, but to get the properties of the universal modality, we will need to have the $S5$ -axioms that hold for U [18] plus the axiom $B\varphi \rightarrow UB\varphi$, which expresses that B is a global notion in a model where U expresses universality. In such a model there is only one center.

There are various possible ways of interpreting the formula $\varphi \succ_B \psi$ in *plausibility models* expressing the belief modality. The foremost question is whether to try to define semantically $\varphi \succ_B \psi$ in terms of the plausibility ordering of the worlds. If one wants to base the strength of belief ordering on the plausibility ordering of the worlds, then immediately the following option comes to mind: the interpretation of $\varphi \succ_B \psi$ can be that there exist φ -worlds which are more plausible than any ψ -world (similar to the proposal in [9]). For $\varphi \succ_B \psi$ it can be given as follows: for each ψ -world there exist φ -worlds which are at least as plausible. If one does this however, $B\varphi$ becomes equivalent to $\varphi \succ_B \neg\perp$. In fact, no distinction in strength of belief can be made between propositions which are believed. This is not at all our aim. A more pleasurable consequence of this definition is that conditional belief $B^\psi\varphi$ can equivalently be expressed as $\neg(\psi \wedge \varphi) \succ_B (\psi \wedge \neg\varphi)$. Ultimately though we think that the price is too high: this interpretation of the ordered formulas gives rise to more unintended validities in the logic $KD45-O$ (cf. section II-C). There are more sophisticated reductions of strengths of beliefs to the plausibility ordering of the worlds but we think that they will all have undesirable consequences. This is only partly due to the fact that belief is identified with truth in the most plausible worlds, which

seems necessary for a dynamic interpretation. To do away with this issue we decided to define another *set-plausibility* ordering \geq_B between the sets of worlds in the *plausibility models*. We have put very minimal requirements on this ordering. In fact, as we will see in section II-B, *belief* can be interpreted in terms of *belief ordering* and *safe beliefs*. How this will relate the world ordering and the set ordering is an interesting study, but we leave it for the future.

We should mention here that the idea of modeling epistemic notions in terms of set orders is not really new. In [19], [20], preferential structures are considered where a preference ordering over worlds is lifted to an ordering of sets of worlds. Plausibility measures are considered in [21] to give a semantics of default logic. These measures can be identified with a partial ordering on sets of worlds and they also provide an interpretation of the notion of beliefs. We are not going into a detailed comparison of these works with ours due to lack of space, but just note that an essential difference is that in our case the ordering of the sets of worlds is only partly determined by the ordering of the worlds.

Alice's belief states (as described in the introduction) can now be formally presented as follows: suppose each of the applicants' names denotes the proposition that "he (she) can do the job". Cora \succ_B Burt in the first case; $(B(\text{Egon}) \wedge \neg B(\text{Deirdre}) \wedge \neg B(\neg \text{Deirdre}))$ implies that Egon \succ_B Deirdre in the second case, with the third case simply being Gregory \succ_B Fiona again, and the fourth one, $B(\neg \text{Harold}) \wedge B(\neg \text{Irma})$. The readers can easily see that in the second case there is some reasoning going on which leads to Egon being given the job, because Alice's belief in the ability of Egon is stronger than her belief in the ability of Deirdre. Even in the fourth example where Alice believes that both Harold and Irma are unable to perform, her belief in the ability of one might be higher than that of the other. Then, if forced to choose, she could do without another round of applications.

A. Axioms and Completeness

Let us first look into some interesting validities of this logic.

- $(\varphi \succ_B \psi) \rightarrow E(\varphi \wedge \neg \psi)$
- $\varphi \succ_B \varphi \wedge \psi$
- $\varphi \vee \psi \succ_B \varphi$
- $B(\varphi \succ_B \psi) \rightarrow (\varphi \succ_B \psi)$

Before providing a complete axiomatization of $KD45-O$, we discuss the motivations behind some of these axioms. Since, by the set-ordering relation in the $KD45-O$ model, \geq_B is a reflexive, transitive and connected relation over $\mathcal{P}(S)$, and \succ_B is the corresponding strict ordering, the following axioms need no introduction:

- $\varphi \succ_B \varphi$ (refl-axiom)
- $(\varphi \succ_B \psi) \wedge (\psi \succ_B \chi) \rightarrow \varphi \succ_B \chi$ (trans-axiom)
- $(\varphi \succ_B \psi) \vee (\psi \succ_B \varphi)$ (lin-axiom)

From these axioms, it follows that,

$$(\varphi \succ_B \psi) \wedge (\psi \succ_B \chi) \rightarrow (\varphi \succ_B \chi)$$

and so we get the transitivity of \succ_B . We have already seen that $E\varphi$ can be defined as $\varphi \succ_B \perp$, because of the 3rd condition that \succ_B satisfies in the model. Using lin-axiom it is easy to show that $U\varphi$ is equivalent to $\perp \succ_B \neg \varphi$. The following axiom takes care of the 2nd condition.

$$(B\varphi \wedge \neg B\psi) \rightarrow (\varphi \succ_B \psi) \quad (\text{center-axiom})$$

Since the ordering formulas are either globally true or globally false in the model, we have:

$$\begin{aligned} (\varphi \succ_B \psi) &\rightarrow B(\varphi \succ_B \psi) && (\text{intros-axiom1}) \\ (\varphi \succ_B \psi) &\rightarrow B(\varphi \succ_B \psi) && (\text{intros-axiom2}) \end{aligned}$$

It immediately follows that,

$$\begin{aligned} \neg(\varphi \succ_B \psi) &\rightarrow B\neg(\varphi \succ_B \psi) \\ \neg(\varphi \succ_B \psi) &\rightarrow B\neg(\varphi \succ_B \psi) \end{aligned}$$

The inverses of all these implications above follow from the lin-axiom. This means that all these ordering statements can be considered to be B -statements, i.e. $\varphi \succ_B \psi$, $\varphi \succ_B \psi$, $U\varphi$, $E\varphi$ are all B -statements. As a result, the inclusion formula concerning the belief and the universal modality, viz. $U\varphi \rightarrow B\varphi$ also follows. The following axiom and rule take care of replacement with equivalent formulas in the ordering formulas:

$$\begin{aligned} U(\varphi \rightarrow \psi) &\rightarrow (\psi \succ_B \varphi) && (U \succ_B \text{-axiom}) \\ \frac{\varphi \rightarrow \psi}{\psi \succ_B \varphi} &&& (\text{inclusion rule}) \end{aligned}$$

We can conclude then

$$\frac{\varphi}{U\varphi} \quad (U \text{gen-rule})$$

and the equivalence rule:

$$\frac{\varphi \leftrightarrow \psi}{\varphi \equiv_B \psi}$$

which means that logically equivalent formulas can be substituted for each other in the ordering formulas as well and hence everywhere. We now have the gen-rule for the universal modality, the K -axiom for U also follows from these principles. We move on to get the other sound ordering axiom, which will help us to get the $S5$ -properties of the universal modality U [18]. It is:

$$\varphi \rightarrow E\varphi \quad (\text{existence axiom})$$

The existence axiom is basically the equivalent ordered formula for $U\varphi \rightarrow \varphi$. We end with the axiom which forces the $KD45-O$ models to have a unique center \mathcal{B} and hence makes B a global property.

$$EB\varphi \rightarrow B\varphi \quad (\text{un.center-axiom})$$

From this the principle $B\varphi \rightarrow UB\varphi$ readily follows. The transitivity and the symmetry axioms for U follow because of the very significant property of $U\varphi$ being a B -statement, the un.center-axiom applies to U -statements as well.

Thus we have the following theorem which is the most basic and important result of this work.

Theorem II.3 *KD45–O is sound and complete with respect to KD45–O models and its validities are completely axiomatized by the following axioms and rules:*

- a) all D axioms and rules
- b) ordering axioms:
 - $\varphi \succ_B \psi$ (refl-axiom)
 - $(\varphi \succ_B \psi) \wedge (\psi \succ_B \chi) \rightarrow \varphi \succ_B \chi$ (trans-axiom)
 - $(\varphi \succ_B \psi) \vee (\psi \succ_B \varphi)$ (lin-axiom)
 - $(B\varphi \wedge \neg B\psi) \rightarrow (\varphi \succ_B \psi)$ (center-axiom)
 - $(\varphi \succ_B \psi) \rightarrow B(\varphi \succ_B \psi)$ (intros-axiom1)
 - $(\varphi \succ_B \psi) \rightarrow B(\varphi \succ_B \psi)$ (intros-axiom2)
 - $U(\varphi \rightarrow \psi) \rightarrow (\psi \succ_B \varphi)$ ($U \succ_B$ -axiom)
 - $\varphi \rightarrow E\varphi$ (existence axiom)
 - $EB\varphi \rightarrow B\varphi$ (un.center-axiom)
- c) inclusion rule:

$$\frac{\varphi \rightarrow \psi}{\psi \succ_B \varphi} \text{ (inclusion rule)}$$

Proof: The readers can easily verify the soundness of these ordering axioms. The $U \succ_B$ -axiom is covered by property 1 of Definition II.2, existence axiom by property 3, and the center axiom describes property 2. With U and E defined as indicated previously, one can easily show that the $S5$ -axioms are derivable for U . It is also not very hard to show completeness using finite sets of sentences.

Assume $\not\models_{KD45-O} \varphi$. We will have to construct a countermodel to φ as a $D-O$ -model. We take a finite adequate set Φ containing φ . In this case an adequate set will be: a set of formulas that is closed under subformulas containing with each formula ψ (a formula equivalent to) $\neg\psi$, containing with $B\psi$ and $B\chi$ (a formula equivalent to) $B(\psi \wedge \chi)$ and a formula (equivalent to) $B(\psi \vee \chi)$. We also need Φ to contain with each formula $B\varphi$ a formula (equivalent to) $UB\varphi$. Finally, Φ contains $B\top$ and $B\perp$. It is easy to see that any finite set is contained in a finite adequate set. We use the Henkin method restricted to Φ . Consider the m.c. (maximally consistent) subsets of Φ . In particular consider such an m.c. set Φ_0 containing $\neg\varphi$. When we now refer to $U\psi$ we mean its translation into $D-O$. It can be shown that the $S5$ -axioms to hold for this translation. The proof is made more perspicuous by referring to U .

The relations \mathcal{R}_B and \mathcal{R}_U are defined as follows:

- $P\mathcal{R}_BQ$ iff (1) for all $B\varphi$ in P , φ as well as $B\varphi$ are in Q ,
(2) for all $\neg B\varphi$ in P , $\neg B\varphi$ in Q .
- $P\mathcal{R}_UQ$ iff (1) for all $U\varphi$ in P , φ as well as $U\varphi$ are in Q ,
(2) for all $\neg U\varphi$ in P , $\neg U\varphi$ in Q .

We have to show that \mathcal{R}_U is an equivalence relation and \mathcal{R}_B a Euclidean sub-relation of \mathcal{R}_U . Finally, within one U -equivalence class there is one, nonempty set of B -reflexive elements, which forms a B -equivalence class. Since all these things are standard we skip this part.

We now take the submodel generated by \mathcal{R}_U from Φ_0 . The set of worlds W of our model will be the set of worlds

in this submodel and the \mathcal{R}_B and \mathcal{R}_U the restrictions of the original \mathcal{R}_B and \mathcal{R}_U to this submodel. \mathcal{R}_U is now the universal relation.

As before, we write \mathcal{B} for the set of \mathcal{R}_B -reflexive elements. The axiom $B\varphi \rightarrow UB\varphi$ implies that this set is unique and a B -equivalence class. The world plausibility ordering is given as follows: any world in \mathcal{B} is more plausible than any in $W \setminus \mathcal{B}$, and within these two sets, the worlds are equi-plausible. So, with respect to the modal operators B and U the model behaves properly, and we have a proper world-ordering as well. We will now have to order $\mathcal{P}(W)$ in a proper way.

Let us say that ψ represents subset X of W if X is the set of nodes where ψ is true, which we may write as $V(\psi) = X$. We say that X is representable if for some $B\psi$ in Φ , ψ represents X . By the conditions on Φ the representable sets are closed under unions and intersections, and contain W itself and the empty set.

The representable subsets of Φ are quasi-linearly ordered by the relation \geq_1 defined by $V(\psi) \geq_1 V(\chi)$ iff $\psi \succ_B \chi$ is true in the model, $V(\psi) >_1 V(\chi)$ iff $\psi \succ_B \chi$ is true in the model. These follow from the first three ordering axioms.

Moreover, if $V(\psi) \subseteq V(\chi)$ then $V(\psi) \geq_1 V(\chi)$ (subset condition), by the axiom: $U(\chi \rightarrow \psi) \rightarrow \psi \succ_B \chi$. Finally if $V(\psi)$ properly contains \mathcal{B} and $V(\chi)$ does not, then $V(\psi) >_1 V(\chi)$ (sufficient belief condition) by the axiom: $B\psi \wedge \neg B\chi \rightarrow \psi \succ_B \chi$.

So, \geq_1 behaves properly on the representable elements of $\mathcal{P}(W)$. What remains is to extend \geq_1 to an ordering \geq with the right properties over all of $\mathcal{P}(W)$.

Take an arbitrary subset X of W . We define $R(X)$ to be the largest subset of X that is representable. That such a set exists follows from the fact that the representable subsets are closed under finite unions and the finiteness of the model.

We now define $X \geq Y$ iff $R(X) \geq_1 R(Y)$. This immediately makes \geq a quasi-linear order. That \geq satisfies the subset condition follows from the fact that, if $X \subseteq Y$, then $R(X) \subseteq R(Y)$.

We will conclude this proof with a lemma showing that \mathcal{B} is representable, i.e. $\mathcal{B} = R(\mathcal{B})$. From that result it follows that, if $\mathcal{B} \subseteq X$, then $\mathcal{B} \subseteq R(X)$. This is clearly sufficient to ensure the sufficient belief condition. So, once we finish the proof of the following lemma, we are done.

Lemma. \mathcal{B} is representable.

Proof of Lemma. Consider w not in \mathcal{B} . Then it is not the case that $w\mathcal{R}_Bw$. This means that, for some particular $B(\psi_w)$ in Φ , $B(\psi_w)$ is in w but ψ_w is not. Note that this implies that ψ_w is true all over \mathcal{B} . Consider the conjunction ψ of all ψ_w for w in the complement of \mathcal{B} . $B(\psi)$ is a member of Φ while ψ is true in all elements of \mathcal{B} , but is falsified at all elements u in the complement of \mathcal{B} , since ψ implies ψ_u and ψ_u is falsified in u . We have shown that \mathcal{B} is represented by ψ . ■

Since the counter-model constructed is finite, we also have that the logic $KD45-O$ is decidable. Before ending this section we mention some intuitively true formulas, which we did not need as axioms, but are definitely worth thinking about. One of them is,

$$(\varphi \succ_B \perp) \rightarrow (\top \succ_B \neg\varphi),$$

which says that if φ is true somewhere then $\neg\varphi$ is not as much to be believed as a tautology. The other direction of the implication can be derived. An equivalent formulation is,

$$(\varphi \succ_B \top) \rightarrow (\perp \succ_B \neg\varphi).$$

To make this true, the model needs an extra clause 4, saying that,

$$\text{if } S \neq X \text{ then } S \succ_B X.$$

This seems a very reasonable addition as it makes the models more symmetric. A more general version of this possible axiom is,

$$(\varphi \succ_B \psi) \rightarrow (\neg\psi \succ_B \neg\varphi).$$

which, if considered definitely increases the already-existing probabilistic flavor of the axiomatization. Another possible principle with a similar flavor is

$$(\varphi \succ_B \psi) \rightarrow (\varphi \wedge \neg\psi) \succ_B (\psi \wedge \neg\varphi).$$

This one exemplifies the notion that if φ is more believed than ψ , then that should be based on the idea that the common part of φ and ψ is irrelevant in the estimation of their relative strengths of belief. Readers can note here that if we strengthen this formula above to its bi-implication, then $(\varphi \succ_B \psi) \rightarrow (\neg\psi \succ_B \neg\varphi)$ follows.

B. Safe Belief

The notion of ‘safe belief’ has been introduced in [17] which also corresponds to “Stalnaker knowledge” [22], where evidence is considered as true information. The authors gave this name to single out those *beliefs* “that are *safe* to hold, in the sense that no future learning of truthful information will force us to revise them.” The safe belief modality is generally denoted by \Box . Evidently, ‘safe beliefs’ are *truthful* ($\Box\varphi \models \varphi$) and *positively introspective* ($\Box\varphi \models \Box\Box\varphi$), but not necessarily *negatively introspective* (in general, $\neg\Box\varphi \not\models \Box\neg\Box\varphi$).

Adding safe belief to our ordering framework is interesting both from the technical as well as intuitive point of view. We already have an understanding of the interplay between beliefs and the comparison of strength of beliefs. Our study will be incomplete, if we do not investigate the lively relationship between the very relevant and important issue of safe beliefs together with our notion of belief orderings.

In the *plausibility models*, the truth definition of $\Box\varphi$ is given by the following clause:

$$\mathcal{M}, s \models \Box\varphi \text{ iff } \mathcal{M}, t \models \varphi \text{ for all worlds } t \leq s.$$

which says that φ can be safely believed at some world s if it holds at all the worlds which are at least as plausible as s . In the following we will introduce the safe belief modality in the setting of $KD45-O$, and give a complete axiomatization of this logic. The language of the logic $KD45-OS$ is defined as follows:

Definition II.4 Given a countable set of atomic propositions Φ , formulas φ are defined inductively:

$$\varphi := \perp \mid p \mid \neg\varphi \mid \varphi \vee \psi \mid B\varphi \mid \Box\varphi \mid \varphi \succ_B \psi$$

where $p \in \Phi$.

We now present the axioms of the logic $KD45-OS$. Together with the axioms and rules of the $KD45$ -logic of beliefs, and the relevant ordering axioms, viz. refl, trans, lin, center, existence, $U \succ_B$ -axiom and the $S4$ -axioms and rules for the safe belief \Box operator, we will have the following extra axioms,

$$\begin{aligned} (\Box\varphi \wedge \neg\Box\psi) &\rightarrow (\varphi \succ_B \psi) && (\Box\text{order-axiom}) \\ (\varphi \succ_B \psi) &\rightarrow \Box(\varphi \succ_B \psi) && (\Box\text{intros-axiom1}) \\ (\varphi \succ_B \psi) &\rightarrow \Box(\varphi \succ_B \psi) && (\Box\text{intros-axiom2}) \end{aligned}$$

In addition to all these, the following axiom relates the operator \Box with B .

$$\Box\varphi \rightarrow B\varphi \quad (\Box B\text{-axiom})$$

The intros-axioms(1-2) and the un.center axioms of $KD45-O$ are derivable from $KD45-OS$. We can also derive:

$$U\varphi \rightarrow \Box\varphi$$

Regarding the \Box order-axiom, it should be mentioned that, unlike belief (center axiom), relating safe belief and belief ordering in this manner may be considered questionable. It says that, if φ is safely believed and ψ is more strongly believed than φ , then ψ can also be safely believed. This seems alright at a first glance, but if we consider the subjectivity of the ordering, this axiom may lead to some dispute. Still, technical reasons make it very desirable, the relationship between the world and set orderings becomes much closer, so we decided to keep this axiom. So, we have the following theorem.

Theorem II.5 The logic $KD45-OS$ is sound and its validities can be completely axiomatized by the following axioms and rules.

- a) all $KD45-O$ axioms and rules
- b) $S4$ -axioms and rules for the modal operator \Box
- c) ordering axioms:
 - $\varphi \succ_B \varphi$ (refl-axiom)
 - $(\varphi \succ_B \psi) \wedge (\psi \succ_B \chi) \rightarrow \varphi \succ_B \chi$ (trans-axiom)
 - $(\varphi \succ_B \psi) \vee (\psi \succ_B \varphi)$ (lin-axiom)
 - $(\perp \succ_B (\neg(\Box\varphi \rightarrow \Box\psi))) \vee (\perp \succ_B \neg(\Box\psi \rightarrow \Box\varphi))$
(\Box lin-axiom)
 - $(B\varphi \wedge \neg B\psi) \rightarrow (\varphi \succ_B \psi)$ (center-axiom)

$$\begin{aligned}
 (\Box\varphi \wedge \neg\Box\psi) &\rightarrow (\varphi \succ_B \psi) && (\Box\text{order-axiom}) \\
 (\varphi \succ_B \psi) &\rightarrow \Box(\varphi \succ_B \psi) && (\Box\text{intros-axiom1}) \\
 (\varphi \succ_B \psi) &\rightarrow \Box(\varphi \succ_B \psi) && (\Box\text{intros-axiom2}) \\
 U(\varphi \rightarrow \psi) &\rightarrow (\psi \succ_B \varphi) && (U \succ_B\text{-axiom}) \\
 \varphi &\rightarrow E\varphi && (\text{existence axiom})
 \end{aligned}$$

$$d) \Box\varphi \rightarrow B\varphi \quad (\Box B\text{-axiom})$$

e) *inclusion rule:*

$$\frac{\varphi \rightarrow \psi}{\psi \succ_B \varphi}$$

We should mention here that, according to [17], belief and conditional belief can be expressed in terms of knowledge and safe belief as,

$$\begin{aligned}
 B^\psi\varphi &:= K\psi \rightarrow K(\psi \wedge \Box(\psi \rightarrow \varphi)), \\
 B\varphi &:= B^\top\varphi,
 \end{aligned}$$

where $K\psi := \neg K\neg\psi$. They gave complete axiomatizations for conditional doxastic logic (logic of conditional belief) as well as the logic of knowledge and safe beliefs. We do not consider knowledge but for this part of the discussion it can be replaced by U . Neither do we talk about conditional belief here, but belief can be defined in terms of the existential modality and safe belief (i.e. in terms of safe belief and belief ordering) as follows:

$$B\varphi := E\Box\varphi$$

Once we have in this manner the modal operator B as a defined concept, we can easily derive all its well-known properties in $KD45-O_S$, but if that holds fully for its relations with \succ_B remains to be seen.

C. Plausibility

Comparing the strength of beliefs explicitly has its various advantageous applications. By plausibility of a proposition we generally mean that we tend to believe in its happening rather than its not happening. That is the interpretation we take here. Hence, in terms of ordered formulas, $P\varphi$ can be expressed as $\varphi \succ_B \neg\varphi$. Of course, there are other possible notions of plausibility, but here we interpret $P\varphi$ as ‘more plausible than not’. We now explore this notion of ‘plausibility’ in terms of belief ordering.

An important principle that will be valid for the *plausibility* operator P is $U(\varphi \rightarrow \psi) \rightarrow (P\varphi \rightarrow P\psi)$. This holds because if $U(\varphi \rightarrow \psi)$, not only will belief in ψ be at least as strong as in φ , but $U(\varphi \rightarrow \psi)$ implies $U(\neg\psi \rightarrow \neg\varphi)$, so belief in $\neg\psi$ is not greater than in $\neg\varphi$. This leads to consequences like $P(\varphi \wedge \psi) \rightarrow P\varphi$.

The reason to take the set semantics for ordering formulas (cf. Definition II.2) becomes clear. If we would adhere to the semantics we may have had for \succ_B in terms of plausibility ordering for worlds (instead of sets of worlds), $P\varphi$ would become equivalent to $B\varphi$, which obviously is undesirable.

One can just subdivide the most plausible worlds (the center) into more and less plausible ones to rectify this, but besides endangering the transition to dynamics this will not yet be really satisfactory in its own right. It will result in interpreting $P\varphi$ into something like ‘ φ is weakly believed’. This would make the modal logic of P a normal modal logic (of weak belief). In particular $P\varphi \wedge P\psi \rightarrow P(\varphi \wedge \psi)$ would become valid, which is not very intuitive.

For example, you may judge it more plausible than not that your next client will be male. Similarly, you may consider it to be plausible that your next client will be a foreigner. But, it doesn’t follow that it is more plausible than not that the next client will be a foreign male, most of one’s foreign clients may be female.

We now move on to showing an independent axiomatization of the plausibility logic P . The language of the P -logic is given by

$$\varphi := p \mid \neg\varphi \mid \varphi \vee \varphi \mid P\varphi$$

We read $P\varphi$ as “ φ is plausible”. As mentioned above, the intuitive meaning of $P\varphi$ can be captured by the formula $\varphi \succ_B \neg\varphi$, and as such, the truth definition of $P\varphi$ in the $KD45-O$ model is given by,

$$\mathcal{M}, s \models P\varphi \text{ iff } \{t \mid \mathcal{M}, t \models \varphi\} \succ_B \{t \mid \mathcal{M}, t \models \neg\varphi\}.$$

Theorem II.6 *P-logic is complete and its validities are completely axiomatized by the following axioms and rules:*

(a) *all propositional tautologies and inference rules*

(b) *plausibility axioms:*

$$P\psi \wedge P\varphi \rightarrow P(\psi \wedge P\varphi)$$

$$P\psi \wedge \neg P\varphi \rightarrow P(\psi \wedge \neg P\varphi)$$

$$P\varphi \rightarrow \neg P\neg\varphi$$

$$P\top$$

(c) *monotonicity rule:*

$$\text{if } \varphi \rightarrow \psi \text{ then } P\varphi \rightarrow P\psi$$

Proof: First of all, we show that any formula in P -logic is equivalent to a formula with P -depth at most one. For that purpose we first derive the following schemes:

$$1) P\psi \rightarrow (\varphi \leftrightarrow \varphi[\top/P\psi])$$

$$2) \neg P\psi \rightarrow (\varphi \leftrightarrow \varphi[\perp/P\psi])$$

Here, $\varphi[\top/P\psi]$ means φ with \top substituted for some occurrences of $P\psi$. We prove by induction on the complexity of formulas φ with possible occurrences of \top and \perp .

In the base case, that is for the atomic propositions and propositional constants, the result follows immediately.

Induction step. This is trivial for the boolean connectives. So, it suffices to prove it for $P\varphi$ assuming it holds for φ . From the induction hypothesis for the first scheme it follows that $(P\psi \wedge \varphi) \leftrightarrow (P\psi \wedge \varphi[\top/P\psi])$ is provable. Now assume $P\psi$ and $P\varphi$. By an axiom $P(\varphi \wedge P\psi)$ follows. From the

fact just proved it follows that $P(\varphi[\top/P\psi] \wedge P\psi)$ and hence $P(\varphi[\top/P\psi])$. The proof for the second scheme is very similar.

To see that these schemes imply that each formula in P -logic is equivalent to a formula with P -depth at most one, just note that $\vdash \varphi \leftrightarrow ((P\psi \wedge \varphi) \vee (\neg P\psi \wedge \varphi))$. Now, if we want to get rid of occurrences of $P\psi$ in φ we can replace φ by $((P\psi \wedge \varphi[\top/P\psi]) \vee (\neg P\psi \wedge \varphi[\perp/P\psi]))$. By doing this consecutively for all occurrences of $P\psi$ with no occurrences of P in ψ we obtain the desired result.

Next, we show that any consistent set has a model. Assume we have a consistent set in the P -logic which can be extended to a maximal consistent set Γ , say. Since we can restrict attention to formulas which are boolean combinations of atoms and formulas of the form $P\varphi$ where φ no longer contains P , a maximal consistent set is essentially only a set of atoms, negations of atoms, such $P\varphi$'s and $\neg P\varphi$'s and their boolean combinations.

We now make a model in our sense where $P\varphi$ gets interpreted as $\varphi >_B \neg\varphi$. The worlds will be simply defined by a number of atoms being true in it and the rest of the atoms false. Let us now consider the following model, $\mathcal{M} = (S, \leq, \geq_B, V)$, where S is the set of all maximal consistent subsets. The ordering of the subsets is as follows: There are 5 equivalence classes in the ordering starting with the highest grade of believability. We take membership of those classes to determine the degree of belief in the sets.

- (1) The whole set, which is of course represented by \top .
- (2) The sets represented by those φ for which $P\varphi$ is in Γ (except for \top).
- (3) The sets represented by those φ for which $\neg P\varphi$ is in Γ as well as $\neg P\neg\varphi$.
- (4) The sets represented by those φ for which $P\neg\varphi$ is in Γ (except for \perp).
- (5) The empty set, which is of course represented by \perp .

These are all possibilities because of axiom $P\varphi \rightarrow \neg P\neg\varphi$. Finally we take \mathcal{B} , the center, to be the whole set (so, there are no beliefs except the trivial one in \top).

The two things we have to check are: First, that, if a set is in class (2), then any larger one will be in (2) as well (or in (1)). This follows from the *monotonicity rule*. Similarly for the other classes. Second, that, if a set X contains all of \mathcal{B} , and another set Y doesn't, then $X > Y$. That is trivial: X has to be \mathcal{B} , the whole set, and Y isn't.

As earlier, we can induce an ordering over all subsets satisfying the required conditions. All the single worlds have to be taken to be equally plausible, i.e. $s \leq t$, for all $s, t \in S$. So, for each consistent set we can have a model in $KD45-O$. So, the axioms and rules given in Theorem 2.6 axiomatize the P -logic of 'more plausible than not'. It is also worth-mentioning why $(P\varphi \wedge P\psi) \rightarrow P(\varphi \wedge \psi)$ will fail in general. There may be sets in (2), the intersection of which, is not in (2). ■

Evidently, $P\varphi$ is a global notion - its value does not vary through the model. Again, P is clearly an introspective notion. Interestingly, the principles 4 and 5 for the modal operator P are derivable in this P -logic, but the K -axiom is not. That $P\varphi \wedge P\psi \rightarrow P(\varphi \wedge \psi)$ ought not to be a valid principle in the P -logic is clear if we interpret $P\varphi$ as $\varphi >_B \neg\varphi$.

Let us finally note that an interpretation of $P\varphi$ as φ as having probability more than 0.5 (or any other number between 0.5 and 1) leads to exactly the P -axioms provided one considers the probability statements themselves to always have probability 1.

We now consider a system having both belief and the plausibility operator, viz. the BP -system. This system will provide pointers to discuss logics of belief and disbelief in the next subsection. The language is that of the P -logic, together with the additional modal operator for belief, B .

$$\varphi := p \mid \neg\varphi \mid \varphi \vee \varphi \mid P\varphi \mid B\varphi$$

Some validities of this logic in the $KD45-O$ model are,

- $B\varphi \rightarrow P\varphi$
- $P\varphi \rightarrow BP\varphi$
- $\neg P\varphi \rightarrow B\neg P\varphi$

Theorem II.7 *BP-logic is complete and its validities are completely axiomatized by the following axioms and rules:*

- a) all propositional tautologies and inference rules
- b) all $KD45$ axioms and rules
- c) all P axioms and rules
- d) special axioms:
 - $B\varphi \rightarrow (\psi \leftrightarrow \psi[\top/B\varphi])$
 - $\neg B\varphi \rightarrow (\psi \leftrightarrow \psi[\perp/B\varphi])$
 - $B\varphi \rightarrow P\varphi$

The proof is very similar to that for the P -logic. It uses the fact that the axioms force all formulas to be equivalent to boolean combinations of atoms and formulas of the form $P\varphi$ and $B\varphi$, where φ is boolean. It is noteworthy that the principle $B\varphi \wedge P\psi \rightarrow P(\varphi \wedge \psi)$ of [23] fails in the BP -logic. It is not difficult to construct a counterexample.

D. Disbelief

Disbelief in a proposition is governed by exactly the opposite situation to the one discussed in the previous subsection, $D\varphi$ can be expressed as $\neg\varphi >_B \varphi$, that is $P\neg\varphi$.

With the huge amount of work going on in logics of beliefs and belief revision, consideration of disbelief as a separate epistemic category came to fore in the latter part of last decade ([24], [25]). Consideration of changing or revising disbeliefs as a process analogous to belief revision was taken up by [26]. Belief-disbelief pairs i.e. simultaneous consideration of belief and disbelief sets were also taken up ([27], [28]) through which various connections of possible

inter-connectivity of beliefs and disbeliefs have come into focus. As mentioned earlier our notion of explicit belief ordering provides another path into expressing the concept of disbelief.

The basic idea for disbelieving a proposition is that, the inclination to believe in its negation is stronger than that to believe it. Consequently, disbelieving is a much weaker notion than believing the negation of the proposition, but it should imply that one does not believe in the proposition. In other words, $D\varphi$ is implied by $B\neg\varphi$ and implies $\neg B\varphi$ but not the other way around in either case.

To exemplify the matter a bit, let us consider the following situation. Due to the unpredictable weather conditions, Pom's belief in that she should not cycle from Amsterdam to Leiden is much stronger than her belief that she should. When options like this are available, it is very natural to have this sort of ordering dilemma playing around people's mind. This can be interpreted as that Pom disbelieves that she should bike, which evidently implies that she does not believe that she should bike. But that 'she believes that she should not bike' is a much stronger statement, which fails to express the finer interplay of doubts that is always prevalent in one's mind.

In general, if a person faces a decision based on whether a certain state of affairs is the case or an event happens, she may not have enough evidence to believe that the state of affairs is the case or is not the case. Then she may base her decision on whether she thinks the state of affairs plausible or disbelieves in it. Only in the case that her strength of belief in the two possibilities is equal, translated into our framework as $\varphi \equiv_B \neg\varphi$, it is a real tossup for her.

Various principles for the 'disbelief' operator together with the 'belief' one have been discussed in [25] in the autoepistemic logic framework of [29]. As such, the possible world semantics provided there which is based on separate sets of worlds for beliefs and disbeliefs is not very interesting, and suffers from 'disjointedness' as well as 'mirror-image' problems. These questions will not arise in the semantics we propose here. The basic reason is the fact that 'disbelief' is given a global stance in contrast to 'belief' which is apparent from their respective interpretations. This also emphasizes the fact that disbelieving something is different from both from 'not believing' as well as 'believing the negation'.

We now focus on getting a more feasible logic of belief and disbelief in similar lines to BP logic introduced earlier. From our formal understanding $D\varphi$ is same as $P\neg\varphi$ and hence we get the following dual axiomatization of the BD -logic -

Theorem II.8 *BD -logic is complete and its validities are completely axiomatized by the following axioms and rules:*

a) *all propositional tautologies and inference rules*

b) *all $KD45$ axioms and rules*

c) *disbelief axioms:*

$$D\varphi \rightarrow (\psi \leftrightarrow \psi[\top/D\varphi])$$

$$\neg D\varphi \rightarrow (\psi \leftrightarrow \psi[\perp/D\varphi])$$

$$D\varphi \rightarrow \neg D\neg\varphi$$

$$D\perp$$

d) *special axioms:*

$$B\varphi \rightarrow (\psi \leftrightarrow \psi[\top/B\varphi])$$

$$\neg B\varphi \rightarrow (\psi \leftrightarrow \psi[\perp/B\varphi])$$

$$B\varphi \rightarrow D\neg\varphi$$

e) *anti-monotone rule:*

$$\text{if } \varphi \rightarrow \psi \text{ then } D\psi \rightarrow D\varphi$$

Some interesting validities of this logic are,

- $B\neg\varphi \rightarrow D\varphi$
- $D\varphi \rightarrow \neg B\varphi$
- $D\varphi \rightarrow BD\varphi$
- $\neg D\varphi \rightarrow B\neg D\varphi$
- $\neg D\varphi \rightarrow DD\varphi$
- $\neg B\varphi \rightarrow DB\varphi$

On the other hand, as in P -logic and BP -logic, the corresponding intuitively incorrect principle, $D\varphi \wedge D\psi \rightarrow D(\varphi \vee \psi)$ can also be avoided in the BD -logic.

E. Preference

There is a very close relationship between an agent's beliefs and her preferences which has been extensively discussed in ([30], [3]). Based on the ideas from *optimality theory*, intrinsic preference on the basis of priority sequences $P_1 \gg \dots \gg P_n$ is formulated. Here, the P_i 's are first-order formulas with exactly one free variable, which is common to all of them. Preferences over objects can be defined in terms of these sequences. The basic idea is to define objective preference by:

$$Pref(d, e) \Leftrightarrow \exists i (P_i d \wedge \neg P_i e) \wedge \forall j < i (P_j d \leftrightarrow P_j e)$$

For subjective preferences over objects, which in fact are considered to be influenced by beliefs, several options are considered. We mention a few of them for the benefit of the readers, their meanings are more or less obvious.

$$Pref(d, e) \Leftrightarrow \exists i (B(P_i d) \wedge \neg B(P_i e) \wedge \forall j < i (B(P_j d) \leftrightarrow B(P_j e)))$$

$$Pref(d, e) \Leftrightarrow \exists i (\neg B(\neg P_i d) \wedge B(\neg P_i e) \wedge \forall j < i (B(\neg P_j d) \leftrightarrow B(\neg P_j e)))$$

$$Pref(d, e) \Leftrightarrow \exists i ((B(P_i d) \wedge \neg B(P_i e)) \vee (\neg B(\neg P_i d) \wedge B(\neg P_i e)) \wedge \forall j < i ((B(P_j d) \leftrightarrow B(P_j e)) \wedge (B(\neg P_j d) \leftrightarrow B(\neg P_j e))))$$

It is clear that the above three approaches are different attempts to express that up to a certain level of the priority sequence the degree of belief in the objects d and e having the mentioned properties is the same and that at the next level the degree of belief in d having the right property is greater than that in e having it. Here we can express this directly in the language as below, and the way greater strength of belief is to be taken in a particular application is then delegated to the semantics.

$$Pref(d, e) \Leftrightarrow \exists i(P_i d \succ_B P_i e \wedge \forall j < i(P_j d \equiv_B P_j e)).$$

F. Multi-agent case

We have been focusing on beliefs and strengths of beliefs of a single agent. The whole idea can be generalized to the multi-agent framework. We only give some preliminary ideas here. The technical details need to be worked out, and we leave it for the future. The language of the logic of belief ordering in the multi-agent case, $D - O_M$ can be defined as follows:

Definition II.9 Given a finite set of agents A , and a countable set of atomic propositions Φ , formulas φ are defined inductively:

$$\varphi := \perp \mid p \mid \neg \varphi \mid \varphi \vee \varphi \mid B_a \varphi \mid \varphi \succ_{B_a} \psi$$

where $p \in \Phi$.

The indices in the belief and universality modality and in the ordering formula denote the agents whose beliefs or strengths of beliefs are considered. The operators \succ_{B_a} and U_a are defined in the usual way. The fact that U is also indexed may surprise the reader for a moment but it is the only coherent way to extend the one agent case. Existence of a location for a proposition to be true meant for us that for the one agent belief in the proposition was stronger than belief in a contradiction. With more agents we may have agents who differ in regard to the existence of propositions: more worlds will have to be added to the model, and it will not stop there: there is no reason for $E_a E_b$ to be equivalent to E_a or E_b , etc. It is appropriate to add a real universality operator U that corresponds to the agent's U_a as the common knowledge operator corresponds to the knowledge of the individual agents. With regard to axioms the upshot is for example that the existence axiom $\varphi \rightarrow (\varphi \succ_{B_a} \perp)$ will have to be weakened to $U_a(\varphi \rightarrow (\varphi \succ_{B_a} \perp))$.

Likewise, the models for $D - O_M$ have to be multi-agent generalizations of those for $D - O$. The basic idea to consider here is that we can no longer rule out worlds that are *impossible* for an agent a . They might well be possible for another agent b and also have to be considered while talking about agent a 's belief about agent b 's beliefs and so on. Evidently, the earlier *plausibility ordering* and *set ordering* of worlds will get indexed by agents (one for each agent), and the global concept of belief will give way to more local concepts of beliefs. This fact becomes apparent in the syntax

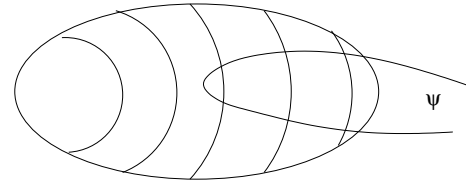


Fig. 2. A public announcement ψ is uttered

also, with the introduction of formulas like $U_a \varphi$. The notion of *comparative classes* [17] which gives the set of worlds that an agent considers relevant while positioned at her current world comes into play. Formally, a comparative class of some world is just the set of worlds that are related to the current world by the plausibility order. To give meaning to agents' beliefs, strength of beliefs, these relevant worlds are needed to be considered only, unlike the single agent case, where the whole model is taken into account. As mentioned earlier, we leave the technical details for later.

III. DYNAMICS OF ORDERING FORMULAS

Till now we have been talking about the static language of belief ordering and its corresponding models, representing the information states (possible worlds) of an agent. We move on to discuss the effect of information-update procedures which change the models under consideration.

It should be mentioned here that we will see an extensive use of the notion of *conditional belief* while discussing the dynamics of belief change. Conditional beliefs [2] *pre-encode* beliefs in a static way in situations which arise after new information comes in. Formulas of the form $B^\psi \varphi$ (believing in φ , given ψ) express that we believe in φ , once we assume that ψ is the case. This induces that if ψ is learnt then it is to be believed that φ was the case (before the learning) [17]. This does not literally tell us what happens after ψ is learnt. For example, if φ is unknown but true and one learns that $\neg B \varphi \wedge \varphi$ then afterwards $B \varphi$ will be true, contrary to what was learnt.

In [2], van Benthem discusses three different update procedures influencing the beliefs of an agent, viz. the influence of *hard* information like *public announcement of ψ* (the not- ψ -worlds get deleted from the model and only the ψ -worlds remain), and of *soft* information with *lexicographic upgrade* (introducing a preference for the ψ -worlds, all the ψ -worlds become more plausible than all the not- ψ -worlds - within these zones, the existing ordering remains), and lastly the impact of soft information with *elite* upgrade (the best ψ -worlds come out on top, beside that the previous ordering remains). We currently just focus on the effect of hard information and soft information with lexicographic upgrade.

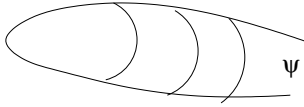


Fig. 3. The influence of hard information

A. Influence of hard information

While discussing the influence of ‘hard’ information over beliefs, [2] considered the following *reduction axioms* for the logic of public announcement in terms of ‘belief’ and ‘conditional belief’, where $[\psi]\varphi$ is to be read as ‘after the public announcement of ψ , φ is true’.

- $[\psi]q \leftrightarrow (\psi \rightarrow q)$
- $[\psi]\neg\varphi \leftrightarrow (\psi \rightarrow \neg[\psi]\varphi)$
- $[\psi](\varphi \wedge \chi) \leftrightarrow ([\psi]\varphi \wedge [\psi]\chi)$
- $[\psi]B\varphi \leftrightarrow (\psi \rightarrow B^\psi[\psi]\varphi)$
- $[\psi]B^x\varphi \leftrightarrow (\psi \rightarrow B^{\psi \wedge [\psi]\chi}[\psi]\varphi)$

In fact, the effect of public announcements of ψ over an *ordered* model should be clear from the figures 2 and 3. Under the announcement of ψ , the earlier model \mathcal{M} , say (cf. fig 2), reduces to a model relativized to ψ (cf. fig 3), which is essentially a submodel of \mathcal{M} , whose domain set is the set where ψ holds.

Let us now first investigate how the *KD45–O* model changes under the influence of a public announcement ψ , say. The definition is as follows:

Definition III.1 A *D–O* model is defined as in definition II.2 as the structure $\mathcal{M} = (S, \leq, \geq_B, V)$. Under the influence of public announcement of ψ the model becomes $\mathcal{M}^{\uparrow\psi} = (S^{\uparrow\psi}, \leq^{\uparrow\psi}, \geq_B^{\uparrow\psi}, V^{\uparrow\psi})$ where $S^{\uparrow\psi} = \{s \in S : \mathcal{M}, s \models \psi\}$, $\leq^{\uparrow\psi} = \leq|_{S^{\uparrow\psi} \times S^{\uparrow\psi}}$, $\geq_B^{\uparrow\psi} = \geq_B|_{\mathcal{P}(S^{\uparrow\psi}) \times \mathcal{P}(S^{\uparrow\psi})}$, and $V^{\uparrow\psi} = V|_{S^{\uparrow\psi}}$.

The model satisfies the following conditions:

- 1) If $X \subseteq Y \subseteq S^{\uparrow\psi}$, then $Y \geq_B^{\uparrow\psi} X$
- 2) if $\mathcal{B}^{\uparrow\psi}$ is the new set of plausible worlds, truth on which suffices to make an assertion to be believed, then $\mathcal{B}^{\uparrow\psi} \subseteq X \subseteq S^{\uparrow\psi} \wedge \mathcal{B}^{\uparrow\psi} \not\subseteq Y \subseteq S^{\uparrow\psi} \Rightarrow X >_B^{\uparrow\psi} Y$, where $>_B^{\uparrow\psi}$ denotes the corresponding strict ordering.
- 3) If X is non-empty, then $X >_B^{\uparrow\psi} \emptyset$.

The truth definitions of the formula $[\psi]\varphi$ in a *KD45–O* model is given by,

$$\mathcal{M}, s \models [\psi]\varphi \text{ iff if } \mathcal{M}, s \models \psi, \text{ then } \mathcal{M}^{\uparrow\psi}, s \models \varphi.$$

Any logic L completing *KD45–O* to contain conditional beliefs will have to contain

$$B^\psi\varphi \wedge \neg B^\psi\chi \rightarrow (\psi \wedge \varphi) \succ_B (\psi \wedge \chi).$$

This axiom ensures that the updated model still has the necessary property 2 of Definition II.2 and thus will be a *KD45–O* model. We have the following theorem:

Theorem III.2 If we take L to be a complete axiomatization of *KD45–O* together with conditional beliefs, then its extension under public announcement is complete and its validities are completely axiomatized by the following axioms and rules in addition to L :

- (a) *PAL reduction axioms for atomic facts, Boolean operations, belief and conditional belief*
- (b) *PAL reduction axioms for ordering formulas:*

$$[\psi](\varphi \succ_B \chi) \leftrightarrow \psi \rightarrow ((\psi \wedge [\psi]\varphi) \succ_B (\psi \wedge [\psi]\chi))$$

$$[\psi](\varphi \succ_B \chi) \leftrightarrow \psi \rightarrow ((\psi \wedge [\psi]\varphi) \succ_B (\psi \wedge [\psi]\chi))$$

B. Influence of soft information

Following [2], we considered the influence of *hard information* over beliefs. Van Benthem continued by discussing the effect of the arrival of *soft information* on beliefs, which does not influence the existence of the possible worlds, only their plausibility ordering. We now give the definition of the changed models under the influence of soft information with lexicographic upgrade and the corresponding reduction formulas.

Definition III.3 A *D–O* model is defined as in definition II.2 as the structure $\mathcal{M} = (S, \leq, \geq_B, V)$. Under the influence of soft information ψ , say the model becomes $\mathcal{M}^{\uparrow\psi} = (S^{\uparrow\psi}, \leq^{\uparrow\psi}, \geq_B^{\uparrow\psi}, V^{\uparrow\psi})$ where $S^{\uparrow\psi} = S$, $\leq^{\uparrow\psi} = \leq|_{K \times K} \cup \leq|_{L \times L} \cup \{(u, v) : u \in K \text{ and } v \in L\}$, where $K = \{s \in S : \mathcal{M}, s \models \psi\}$, $L = S \setminus K$, $\geq_B^{\uparrow\psi} = \geq_B$, and $V^{\uparrow\psi} = V$.

Once again, the model $\mathcal{M}^{\uparrow\psi}$ satisfies the conditions,

- 1) If $X \subseteq Y$, then $Y \geq_B^{\uparrow\psi} X$
- 2) if $\mathcal{B}^{\uparrow\psi}$ is the new set of plausible worlds, truth on which suffices to make an assertion to be believed, then $\mathcal{B}^{\uparrow\psi} \subseteq X \wedge \mathcal{B}^{\uparrow\psi} \not\subseteq Y \Rightarrow X >_B^{\uparrow\psi} Y$, where $>_B^{\uparrow\psi}$ denotes the corresponding strict ordering.
- 3) If X is non-empty, then $X >_B^{\uparrow\psi} \emptyset$.

The truth definitions of the formula $[\psi]\varphi$ in a *KD45–O* model is given by,

$$\mathcal{M}, s \models [\uparrow\psi]\varphi \text{ iff } \mathcal{M}^{\uparrow\psi}, s \models \varphi.$$

We have the following theorem:

Theorem III.4 If we take L to be a complete axiomatization of *KD45–O* with conditional beliefs, then its extension under announcement of soft information with lexicographic upgrade is complete and its validities are completely axiomatized by the following axioms and rules in addition to L :

- (a) *reduction axioms:*

$$[\uparrow\psi]q \leftrightarrow q$$

$$[\uparrow\psi]\neg\varphi \leftrightarrow \neg[\uparrow\psi]\varphi$$

$$[\uparrow\psi](\varphi \wedge \chi) \leftrightarrow ([\uparrow\psi]\varphi \wedge [\uparrow\psi]\chi)$$

$$[\uparrow\psi]B^x\varphi \leftrightarrow (E(\psi \wedge [\uparrow\psi]\chi) \wedge (B^{\psi \wedge [\uparrow\psi]\chi}[\uparrow\psi]\varphi \vee B^{[\uparrow\psi]\chi}[\uparrow\psi]\varphi))$$

$$\begin{aligned}
 [\uparrow \psi](\varphi \succ_B \chi) &\leftrightarrow E\psi \wedge (((\psi \wedge [\uparrow \psi]\varphi) \succ_B (\psi \wedge [\uparrow \psi]\chi)) \vee ([\uparrow \psi]\varphi \succ_B [\uparrow \psi]\chi)) \\
 [\uparrow \psi](\varphi \succ_B \chi) &\leftrightarrow E\psi \wedge (((\psi \wedge [\uparrow \psi]\varphi) \succ_B (\psi \wedge [\uparrow \psi]\chi)) \vee ([\uparrow \psi]\varphi \succ_B [\uparrow \psi]\chi))
 \end{aligned}$$

The product-update model [2] or the more general version of the action-priority update model [17] may not be of much use in the modelling of the ordering formulas, because the first order frame conditions of $KD45 - O$ -models are not all universal Horn sentences (e.g. the linearity condition), which is required for easy passage to this type of update models [2].

IV. CONCLUSION AND FURTHER WORK

An explicit ordering of formulas to compare the strengths of beliefs is introduced. A complete axiomatization for this belief logic with explicit ordering is provided. This notion aids in giving intuitive formulations for various related concepts like universality as well as some other epistemic attitudes - much older and thoroughly discussed notions like *universality* and *preference*, together with relatively newer ones like *plausibility* and *disbelief*. Independent axiomatizations for the logics of plausibility, belief and plausibility as well as belief and disbelief are also provided. Interplay of belief ordering with the concept of safe beliefs is discussed. Lastly, we delved into the dynamics of this ordering concept, e.g. the effect of hard as well as soft information over the ordering formulas. A few possible avenues for future work are discussed below.

a) *Interpreting the ordered formulas*: We have provided different ways of interpreting these ordered belief formulas, and a complete axiomatization is provided with respect to a most general one. It will be interesting to find extensions for other possible applications and their properties, specially to get more interesting inter-connections between the world-ordering and the set-ordering.

b) *Dynamic setting*: As evident from the discussions in section 3, the whole system fits very well into the dynamic epistemic logic framework. But clearly this is the just the start of serious research in this area. There are various notions in this context that need to be thoroughly investigated, especially the relation to conditional belief. There are also important model-theoretic issues that one can look into. In short, a lot of possibilities have emerged with the introduction of this ordering of beliefs in the already existing dynamic logic framework dealing with epistemic attitudes.

Acknowledgements: We thank the unknown referees for their close reading and extensive comments. We will keep these comments in mind when writing a more extensive journal version. The second author also thanks the Center for Soft Computing Research, Indian Statistical Institute, Kolkata for the congenial atmosphere she had there while doing this work during February, 2008 to February, 2009 and also August, 2009.

REFERENCES

- [1] J. Gerbrandy, "Bisimulation on planet kripke," Ph.D. dissertation, University of Amsterdam, 1999.
- [2] J. v. Benthem, "Dynamic logic for belief revision," *Journal of Applied Non-Classical Logic*, vol. 17, no. 2, pp. 129–155, 2007.
- [3] F. Liu, "Changing for the better: Preference dynamics and agent diversity," Ph.D. dissertation, University of Amsterdam, 2008.
- [4] S. Artemov and L. Beklemishev, "Provability logic," in *Handbook of Philosophical Logic*, 2nd ed., D. Gabbay and F. Guentner, Eds. Kluwer, Dordrecht, 2004, vol. 13.
- [5] D. Guaspari and R. Solovay, "Rosser sentences," *Annals of Mathematical Logic*, vol. 16, pp. 81–89, 1979.
- [6] D. d. Jongh, "A simplification of a completeness proof of Guaspari and Solovay," *Studia Logica*, vol. 46, pp. 187–192, 1987.
- [7] K. Segerberg, "Qualitative probability in a modal setting," in *Proceedings of the 2nd Scandinavian Logic Symposium*, J. Fenstad, Ed. Amsterdam: North-Holland, 1971.
- [8] P. Gärdenfors, "Qualitative probability as an intentional logic," *Journal of Philosophical Logic*, vol. 4, pp. 171–185, 1975.
- [9] D. Lewis, *Counterfactuals*. Blackwell and Harvard U.P., 1973.
- [10] W. Spohn, "Ordinal conditional functions. a dynamic theory of epistemic states," in *Causation in Decision, Belief Change, and Statistics*, W. Harper and B. Skyrms, Eds. Kluwer, Dordrecht, 1988, vol. II.
- [11] O. Board, "Dynamic intractable epistemology," *Games and Economic Behavior*, vol. 49, pp. 49–80, 2004.
- [12] P. Gärdenfors and D. Makinson, "Revisions of knowledge systems and epistemic entrenchment," in *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge*, M. Vardi, Ed. Los Altos: Morgan Kaufmann, 1988, pp. 83–95.
- [13] G. v. Wright, "Deontic logic," *Mind*, vol. 60, pp. 1–15, 1951.
- [14] J. Hintikka, *Knowledge and Belief*. Ithaca, N.Y.: Cornell University Press, 1962.
- [15] S. Kripke, "Semantical considerations on modal logics," *Acta Philosophica Fennica*, vol. 16, pp. 83–94, 1963.
- [16] J. Halpern and Y. Moses, "A guide to completeness and complexity for modal logics of knowledge and belief," *Artificial Intelligence*, vol. 54, pp. 319–379, 1992.
- [17] A. Baltag and S. Smets, "A qualitative theory of dynamic interactive belief revision," in *Logic and the Foundations of Game and Decision Theory, Texts in Logic and Games*, G. Bonanno, W. van der Hoeck, and M. Wooldridge, Eds., vol. 3. Amsterdam University Press, 2008, pp. 9–58.
- [18] V. Goranko and S. Passy, "Using the universal modality: Gains and questions," *Journal of Logic and Computation*, vol. 2, no. 1, pp. 5–30, 1992.
- [19] J. Halpern, "Defining relative likelihood in partially-ordered preferential structures," *Journal of AI Research*, vol. 7, pp. 1–24, 1997.
- [20] —, *Reasoning About Uncertainty*. MIT Press, 2003.
- [21] N. Friedman and J. Halpern, "Plausibility measures and default reasoning," *Journal of the ACM*, vol. 48, no. 4, pp. 648–685, 2001.
- [22] R. Stalnaker, "On logics of knowledge and belief," *Philosophical Studies*, vol. 128, no. 1, pp. 169–199, 2006.
- [23] J. Burgess, "Probability logic," *Journal of Symbolic Logic*, vol. 34, no. 2, pp. 264–274, 1969.
- [24] A. Ghose and R. Goebel, "Belief states as default theories: Studies in non-prioritised belief change," in *Proceedings of the 13th European Conference on Artificial Intelligence*, H. Prade, Ed., 1998, pp. 8–12.
- [25] A. Gomolinska, "On the logic of acceptance and rejection," *Studia Logica*, vol. 60, pp. 233 – 251, 1998.
- [26] A. Gomolinska and D. Pearce, "Disbelief change," *Electronic essays on the occasion of the fiftieth birthday of Peter Gärdenfors*, 2001.
- [27] S. Chopra, J. Heidema, and T. Meyer, "Logics of belief and disbelief," in *Proceedings of the ninth International Workshop on Non-Monotonic Reasoning*, 2002.
- [28] M. Chakraborty and S. Ghosh, "Belief-disbelief interface: A bi-logical approach," *Fundamenta Informaticae*, to appear.
- [29] R. Moore, "Semantical considerations on nonmonotonic logic," *Artificial Intelligence*, vol. 25, pp. 75–94, 1985.
- [30] D. d. Jongh and F. Liu, "Optimality, belief and preference," in *Proceedings of the Workshop on Rationality and Knowledge, ESSLLI*, S. Artemov and R. Parikh, Eds., 2006.

Knowledge about lights along a line

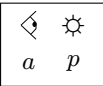
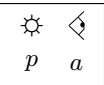
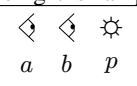
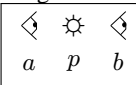
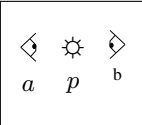
François SCHWARZENTRUBER

Abstract—In this article, we are going to talk about spatial situations. Every agent (human, camera etc.) and every proposition (lamp, object, etc.) are located in the space (here a line) and we express properties over a situation using standard epistemic logic language possibly extended with public announcements. We study links between validities of this geometrical version of epistemic logic and the standard one. We also investigate complexities of model checking and satisfiability.

Keywords: Multi-agent system. Epistemic logic. Spatial reasoning. Public announcements. Pedagogical tool. Complexity theory. Polynomial hierarchy.

I. INTRODUCTION

Many authors in logic and in Artificial Intelligence [5] developed epistemic logic and studied mathematical properties of it. Epistemic logic is theoretical and may be difficult to explain to students. This is the reason why in this article we are going to study a concrete example of multi-agent system. Let us take a line. We are going to put lamps and agents on this line as shown in the Figure 1. Now the question is “what do agents know about lamps and knowledge of other agents about lamps”? This system has been implemented as a pedagogical tool in order to illustrate any epistemic logic course. Indeed, students can easily understand some epistemic logic on concrete examples:

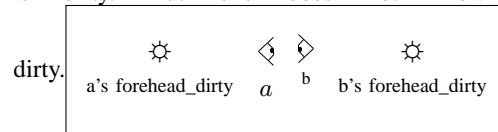
- Agent a sees the lamp p on, so he knows p ; 
- Agent a does not see the lamp p , so he does not know whether p or $\neg p$; 
- Agent a sees another agent b seeing the lamp p on, so agent a knows agent b knows p ; 
- Agent a sees another agent b , and the lamp p , but agent a sees that agent b is not looking in the direction of the lamp p , so agent a knows agent b does not know whether p or $\neg p$; 
- Agent a and agent b are looking at each other and there is the lamp p between them, so there is common knowledge that p is true. This kind of situations have already been considered in [8]. 

This approach can be compared to the approach in [2] for first order logic. In [2], you put objects like cube, pyramids and you can then write formulas in first order logic to check properties over those objects. Here the approach is similar:

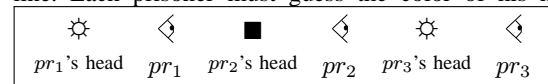
you put agents and lamps and then you can write formulas in epistemic logic to check whether properties over those agents and lamps are true.

Generally speaking many examples of epistemic situation mix time and space. The link between time and knowledge (perfect recall etc.) has been studied and you can find a survey in [3]. There exists also some work linking space and knowledge like in [10]: they provide a logic with a spatial modal operator dealing with topology and an epistemic modal operator. Here, our approach is different: we want to deal with a spatially grounded epistemic logic. We are not going to provide operators in the language to deal with space but only provide an epistemic operator for each agent in the language. The semantics will then directly rely on the geometrical properties of a line. We would like to describe a situation but directly by the graphical and natural representation of the system and not with a Kripke structure. We can formalize well known toy examples as Russian cards [13], Muddy children ([4], [11], [3]) or the prisoner’s test. For instance:

- The Muddy children. The spatial configuration is the following: two children are looking at each other. One knows the other’s forehead is dirty. But one does not know he is



- The prisoner’s test. There are three prisoners on a line. Each prisoner must guess the color of his head.



Another motivation would be video surveillance. Propositions are objects we have to take care of. Agents are camera. We then can specify the video surveillance with epistemic logic formulas. Another possible application may be robots’ space and knowledge reasoning because robots evolve in our spatial world. A last application could be video games. In many role playing games or strategy games, players or non-playing characters can have knowledge about the virtual world. The behaviours of a non-playing character can then be described by the game designer using a knowledge based programming language. For instance, the designer can specify that the guardian of the castle gets crazy if he knows that the door of the castle is open. A preliminary work about formalizing the video game Thief has been done in [7].

A piece of software is available on the Web Site <http://www.irit.fr/~Francois.Schwarzentruber/agentsandlamps/>. It provides a model-checker: you specify the graphical situation and a formula written using epistemic modal operators and/or public announcements operators. In this article:

- We are going to present the semantics of the geometric version of epistemic logic in section II;
- We are going to deal with the model checking and satisfiability problems' complexities in section III;
- In section IV, we will add public announcements to our language to model examples like Muddy children;
- In section V, we are going to present the current implementation.

II. SEMANTICS

We are going to define a new logic based on the same language than the epistemic logic $S5_n$ [5]. $S5_n$ is the logic of frames where relations are equivalence relations. Here we are defining a logic where the semantics is based on a geometric point of view.

A. Language

Our logic is based on the same language as $S5_n$'s one. Let us recall the language of the epistemic logic $S5_n$ [5].

Definition 1 (language):

Let ATM be a countable set of atomic propositions. Let AGT be a countable set of agents. The language \mathcal{L}_{AGT} is defined by the following BNF:

$$\varphi ::= \top \mid p \mid \varphi \wedge \varphi \mid \neg\varphi \mid K_a\psi$$

where $p \in ATM$ and $a \in AGT$.

As usual, $\varphi \vee \psi \stackrel{def}{=} \neg(\neg\varphi \wedge \neg\psi)$. $\hat{K}_a\psi \stackrel{def}{=} \neg K_a\neg\psi$.

Notice that we can only deal with knowledge (operator K_a) and states of lamps (proposition p is true means that the lamp called p is on) in the language. One may expect to deal also with position of lamps, position of agents or maybe spatial topologic operator like in [10] etc. This may be very interesting, especially in all applications cited in the introduction. For instance, we can not express a sentence like "the guardian knows that the beetle is *near* the old man." but we can say "The guardian knows that the beetle knows the old man's hat is red." ($K_{guardian}K_{beetle}old_man_red$) Here we have preferred to keep the language of classical epistemic logic for two reasons:

- a pedagogical tool for understanding epistemic logic should be simple and should have a simple syntax;
- to focus on complexity results with the simple expressivity as $S5_n$.

B. Definitions

The semantics is not defined with a class of models but directly from what a concrete situation is. From this, we will obtain a spatially grounded epistemic logic. A *world* is situation where all agents have a *location* (position and direction where they look), all *lamps* (atomic propositions) have a *location* and a *state* (on or off). Formally:

Definition 2 (world):

A *world* w is a tuple $\langle p_{AGT}, d_{AGT}, p_{ATM}, \pi \rangle$ where:

- $p_{AGT} : AGT \rightarrow \mathbb{R}$;
- $d_{AGT} : AGT \rightarrow \{-1, +1\}$;
- $p_{ATM} : ATM \rightarrow \mathbb{R}$;

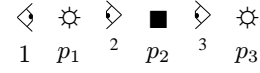


Fig. 1. Example of a world

- $\pi : ATM \rightarrow \{\perp, \top\}$.

The set of all worlds is noted W .

In a world $\langle p_{AGT}, d_{AGT}, p_{ATM}, \pi \rangle$, $p_{AGT}(a)$ denotes the position of agent a . $d_{AGT}(a)$ denotes the direction where the agent a looks: if $d_{AGT}(a) = +1$, the agent a will look on the right and if $d_{AGT}(a) = -1$, he will look on the left. $p_{ATM}(p)$ denotes the position of the lamp saying whether p is true or not. $\pi(p) = \top$ iff the lamp "p" is on. $\pi(p) = \perp$ means that the lamp "p" is off.

We have defined a world in the more close to the reality manner: that is to say using the real numbers. We could also consider locations of agents and lamps as a total preorder over $ATM \cup AGT$. Considering a total preorder is discussed at the end of this section and total preorder is used in Section III. Here we prefer to use the Definition 8 whose advantage is that it can be easily generalized to dimension $n \geq 2$: you just have to replace \mathbb{R} by \mathbb{R}^n and to adapt the notion of direction. In dimension 2 or more, total preorders can no longer be used.

We can also discuss the Definition 8 by the way propositions are treated. Here, a proposition p is associated to a point $p_{ATM}(p)$. This seems to be the simple way to define the semantics. But be aware that in some cases this is a limitation:

- Maybe a proposition p can be associated to a set of points. For instance, if you are at home, you can know it rains either by looking towards the window of the left L or the window of the right R . Hence, here the proposition *rain* may be associated to the set of points $\{L, R\}$;
- Maybe you want that a lamp is associated not to a proposition but more generally to a formula. For instance, when you know that the alarm system located on point P is on, you in fact know that either there is an oil problem or overheating. Hence, here the point is associated to the formula $oil_problem \vee overheating$.

Here we stay with the simple definition for two reasons:

- it is easier for a pedagogical tool to have a simple and clear semantics;
- it is easier for us to begin study a simple case.

Definition 3 (cone):

Let us consider a world $w = \langle p_{AGT}, d_{AGT}, p_{ATM}, \pi \rangle$. We note $cone(a)$ the set $\{p_{AGT}(a) + \lambda \cdot d_{AGT}(a) \mid \lambda \in \mathbb{R}^+\}$.

$cone(a)$ denotes all the set of points the agent a sees.

Example 1: The Figure 1 gives us an example of a world w . We have:

- $p_{AGT}(1) = 0; p_{AGT}(2) = 2; p_{AGT}(3) = 4$;
- $d_{AGT}(1) = +1; d_{AGT}(2) = -1; d_{AGT}(3) = -1$;
- $p_{ATM}(p_1) = 1; p_{ATM}(p_2) = 3; p_{ATM}(p_3) = 5$;
- $\pi(p_1) = \top; \pi(p_2) = \perp; \pi(p_3) = \top$;
- $cone(1) = [0, +\infty[$;
- $cone(2) =]-\infty, 2]$;

- $\text{cone}(3) =] - \infty, 4]$.

Now we are going to define the epistemic relation over worlds. $wR_a u$ means that agent a can not distinguish w from u . In other words, $wR_a u$ iff agent a sees the same things in w and u . Formally:

Definition 4 (epistemic relation):

Let $a \in \text{AGT}$. We define the relation R_a over worlds: $\langle p_{\text{AGT}}, d_{\text{AGT}}, p_{\text{ATM}}, \pi \rangle R_a \langle p'_{\text{AGT}}, d'_{\text{AGT}}, p'_{\text{ATM}}, \pi' \rangle$ iff for all $b \in \text{AGT}$, for all $p \in \text{ATM}$,

- if $p_{\text{AGT}}(b) \in \text{cone}(a)$ then $p_{\text{AGT}}(b) = p'_{\text{AGT}}(b)$ and $d_{\text{AGT}}(b) = d'_{\text{AGT}}(b)$;
- if $p_{\text{AGT}}(b) \notin \text{cone}(a)$ then $p'_{\text{AGT}}(b) \notin \text{cone}(a)$;
- if $p_{\text{ATM}}(p) \in \text{cone}(a)$ then $p_{\text{ATM}}(p) = p'_{\text{ATM}}(p)$ and $\pi(p) = \pi'(p)$
- if $p_{\text{ATM}}(p) \notin \text{cone}(a)$ then $p'_{\text{ATM}}(p) \notin \text{cone}(a)$.

Briefly, suppose that $wR_a u$. If agent a see the agent b in the world w , then he will also see agent b in world u and agent b will have the same location (position and direction). If agent a does not see agent b in the world w , then he also does not see agent b in u . If agent a see the lamp p in the world w , then he will also see the lamp p in world u . The lamp will have the same position and state both in w and u . If agent a does not see the lamp p in w , then he will also not see the lamp p in u .

Until now, we have finally defined a model $\mathcal{M} = \langle W, (R_a)_{a \in \text{AGT}}, \nu \rangle$ where ν maps each world $w \in W$ to π_w . From now, the truth conditions is standard:

Definition 5 (truth conditions):

Let $w \in W$. We define $w \models \varphi$ by induction:

- $w \models \top$;
- $w \models p$ iff $\pi(p) = \top$
- $w \models \varphi \wedge \psi$ iff $w \models \varphi$ and $w \models \psi$;
- $w \models \neg \varphi$ iff $w \not\models \varphi$;
- $w \models K_a \psi$ iff for all w' , $wR_a w'$ implies $w' \models \psi$.

C. Comparison with epistemic logic

Now we are going to compare the epistemic logic $S5_n$ and the set of validities we obtain with the truth conditions of Definition 12. First we give the definition of validities.

Definition 6 (set of validities):

We denote the set of all validities by L^{*1D} , that is to say, $L^{*1D} = \{\varphi \in \mathcal{L}_{\text{AGT}} \mid \forall w \in W, w \models \varphi\}$.

In " L^{*1D} ", " $1D$ " stands for "one dimension" (a line). Now, we can see that our set L^{*1D} contains all validities of $S5_n$.

Proposition 1: $S5_n \subseteq L^{*1D}$.

Proof: We prove that for all $a \in \text{AGT}$, the relation R_a is an equivalence relation. Hence, the model \mathcal{M} is a model of the logic $S5_n$ and satisfies validities of $S5_n$. We have to prove reflexivity, symmetry and transitivity. Let us just begin to prove transitivity. Suppose we have: $\langle p_{\text{AGT}}, d_{\text{AGT}}, p_{\text{ATM}}, \pi \rangle R_a \langle p'_{\text{AGT}}, d'_{\text{AGT}}, p'_{\text{ATM}}, \pi' \rangle$ and $\langle p'_{\text{AGT}}, d'_{\text{AGT}}, p'_{\text{ATM}}, \pi' \rangle R_a \langle p''_{\text{AGT}}, d''_{\text{AGT}}, p''_{\text{ATM}}, \pi'' \rangle$. Let us prove that $\langle p_{\text{AGT}}, d_{\text{AGT}}, p_{\text{ATM}}, \pi \rangle R_a \langle p''_{\text{AGT}}, d''_{\text{AGT}}, p''_{\text{ATM}}, \pi'' \rangle$.

First we have $p_{\text{AGT}}(a) \in \text{cone}(a)$. So $p_{\text{AGT}}(a) = p'_{\text{AGT}}(a) = p''_{\text{AGT}}(a)$ and $d_{\text{AGT}}(a) = d'_{\text{AGT}}(a) = d''_{\text{AGT}}(a)$. In other words, $\text{cone}(a) = \text{cone}'(a) = \text{cone}''(a)$.

From now on, if $p_{\text{AGT}}(b) \in \text{cone}(a)$, then $p_{\text{AGT}}(b) = p'_{\text{AGT}}(b)$ and $d_{\text{AGT}}(b) = d'_{\text{AGT}}(b)$. But, we have effectively, $p_{\text{AGT}}(b) = p'_{\text{AGT}}(b)$ and $\text{cone}(a) = \text{cone}'(a)$. So $p'_{\text{AGT}}(b) \in \text{cone}'(a)$. So $p'_{\text{AGT}}(b) = p''_{\text{AGT}}(b)$ and $d'_{\text{AGT}}(b) = d''_{\text{AGT}}(b)$. Finally, $p_{\text{AGT}}(b) = p''_{\text{AGT}}(b)$ and $d_{\text{AGT}}(b) = d''_{\text{AGT}}(b)$. The other cases are treated in the same manner. ■

The semantics of $K_a p$ in L^{*1D} corresponds to the fact that the agent a sees the light p and the light p is on. More generally, $K_a \psi$ means that the agent a has the proof that ψ . That is why we have those validities in L^{*1D} :

Proposition 2: Let $p, q \in \text{ATM}$.

$\models_{L^{*1D}} K_1(p \vee q) \rightarrow K_1 p \vee K_1 q$.

$\models_{L^{*1D}} K_1(\neg p \vee \neg q) \rightarrow K_1 \neg p \vee K_1 \neg q$.

If $p \neq q$, $\models_{L^{*1D}} K_1(p \vee \neg q) \rightarrow K_1 p \vee K_1 \neg q$

Proof: Let us prove $\models_{L^{*1D}} K_1(p \vee q) \rightarrow K_1 p \vee K_1 q$.

Let $w = \langle p_{\text{AGT}}, d_{\text{AGT}}, p_{\text{ATM}}, \pi \rangle$ be a world such that $w \models K_1(p \vee q)$. We are going to prove that either $w \models K_1 p$ or $w \models K_1 q$. We have $p_{\text{ATM}}(p) \in \text{cone}(1)$ or $p_{\text{ATM}}(q) \in \text{cone}(1)$. Indeed, if we suppose the contrary, that is to say $p_{\text{ATM}}(p) \notin \text{cone}(1)$ and $p_{\text{ATM}}(q) \notin \text{cone}(1)$, there exists a world $u = \langle p_{\text{AGT}}, d_{\text{AGT}}, p_{\text{ATM}}, \pi' \rangle$ such that $\pi'(p) = \perp$ and $\pi'(q) = \perp$ and $wR_1 u$. Hence, $w \not\models K_1(p \vee q)$. Contradiction. So $p_{\text{ATM}}(p) \in \text{cone}(1)$ or $p_{\text{ATM}}(q) \in \text{cone}(1)$. For instance, $p_{\text{ATM}}(p) \in \text{cone}(1)$. And for all $u \in R_1(w)$, $\pi_u(p) = \top$. So $w \models K_1 p$. The other cases are treated in the same manner. ■

Informally, $K_1(p \vee q)$ means that agent 1 has a proof that $p \vee q$. In other words, either he sees p on, or he sees q on. Hence, either $K_1 p$ or $K_1 q$. Nevertheless, $K_1(\varphi \vee \psi) \rightarrow K_1 \varphi \vee K_1 \psi$ is not valid in L^{*1D} .

Notice that there are crucial differences between $S5_n$ and L^{*1D} :

- $S5_n$ is defined as the logic of a class of frames and has the property of uniform substitution. If $\models_{S5_n} \varphi[p]$, we have $\models_{S5_n} \varphi[\psi/p]$ for every formula $\psi \in \mathcal{L}_{\text{AGT}}$;
- On the contrary (see Definition 12), L^{*1D} is defined as the set of formulas valid on *one* model: the model \mathcal{M} . As the definition of R_a (Definition 4) depends on worlds, and especially on valuations, it is not surprising that L^{*1D} does not have the property of uniform substitution. A just *one* model semantics may seem a poor pedagogical application. But, the model \mathcal{M} is big (if AGT and ATM are finite, the size of \mathcal{M} is exponential in $\text{card}(\text{ATM} \cup \text{AGT})$). In fact, you can imagine the model \mathcal{M} to be a kind of canonical model. The model \mathcal{M} is made up with many connected components. For instance, Figure 2 and 8 show two connected components of the model \mathcal{M} .

Now, here is a Proposition showing that we can have common knowledge only when $K_1 K_2 p \wedge K_2 K_1 p$.

Proposition 3: We have:

$\models_{L^{*1D}} K_1 K_2 p \wedge K_2 K_1 p \rightarrow K_1 K_2 K_1 \dots K_2 \dots p$ where " $K_1 K_2 K_1 \dots K_2 \dots$ " denotes any finite sequence of K_1 and K_2 .

Proof: Let $w = \langle p_{\text{AGT}}, d_{\text{AGT}}, p_{\text{ATM}}, \pi \rangle$ be world such that $w \models K_1 K_2 p \wedge K_2 K_1 p$. We want to prove that $w \models K_1 K_2 K_1 \dots K_2 \dots p$. We are going to prove that:

- $p_{\text{AGT}}(2) \in \text{cone}(1)$;

- $p_{AGT}(1) \in cone(2)$;
- $p_{ATM}(p) \in cone(1)$;
- $p_{ATM}(p) \in cone(2)$.

Let us prove $p_{ATM}(p) \in cone(1)$ by contradiction. Suppose that $p_{ATM}(p) \notin cone(1)$. Thus there exists a world $w' = \langle p'_{AGT}, d'_{AGT}, p'_{ATM}, \pi' \rangle$ such that $w R_1 w'$ and $\pi'(p) = \perp$.

We have $w' \not\models p$ so $w' \not\models K_2 p$. So $w \not\models K_1 K_2 p$. Contradiction.

Same proof for $p_{ATM}(p) \in cone(2)$.

Let us prove that $p_{AGT}(2) \in cone(1)$ by contradiction. Suppose that $p_{AGT}(2) \notin cone(1)$. Thus there exists a world $w' = \langle p'_{AGT}, d'_{AGT}, p'_{ATM}, \pi' \rangle$ such that $w R_1 w'$ and $d'_{AGT}(2)$ is such that $p_{ATM}(p) \notin cone'(2)$. Thus, there exists a world $w'' = \langle p''_{AGT}, d''_{AGT}, p''_{ATM}, \pi'' \rangle$ such that $w' R_2 w''$ and $\pi''(p) = \perp$. So $w' \not\models K_2 p$. Hence $w \not\models K_1 K_2 p$. Contradiction.

Same proof for $p_{AGT}(1) \in cone(2)$.

Now we can prove by induction on n that for $n \in \mathbb{N}$, for all $u \in (R_1 \circ R_2)^n(w)$, we have:

- $p_{AGT}(2) \in cone(1)$;
- $p_{AGT}(1) \in cone(2)$;
- $p_{ATM}(p) \in cone(1)$;
- $p_{ATM}(p) \in cone(2)$.
- $\pi(p) = \top$.

Hence $w \models K_1 K_2 K_1 \dots K_2 \dots p$. ■

The validity $K_1 K_2 p \wedge K_2 K_1 p \rightarrow K_1 K_2 K_1 \dots K_2 \dots p$ expresses that if $K_1 K_2 p \wedge K_2 K_1 p$ then the state of the lamp p is the topic of a *mutual social perception*, studied in [8].

Corollary 1: If $n \geq 2$ or $card(ATM) \geq 2$, $S5_n \subsetneq L^{*1D}$.

Proof: The formula $K_1(p \vee q) \rightarrow K_1 p \vee K_1 q$ and $K_1 K_2 p \wedge K_2 K_1 p \rightarrow K_1 K_2 K_1 p$ are in L^{*1D} but are not valid in $S5_n$. ■

More surprising is the fact that common knowledge is not guaranteed by $K_1 K_2 \varphi \wedge K_2 K_1 \varphi$ for all φ . More precisely, $K_1 K_2 \varphi \wedge K_2 K_1 \varphi \rightarrow K_1 K_2 K_1 \varphi$ is not L^{*1D} -valid for all φ . Look at the model of the Figure 2: agent 1 = agent in blue. agent 2 = agent in red. Consider the world on the bottom on the right. Let us call it w . We have $w \models K_1 K_2 \neg K_2 p \wedge K_2 K_1 \neg K_2 p$. But, we have $w \not\models K_1 K_2 K_1 \neg K_2 p$. Indeed there exists w' such that $w R_1 \circ R_2 \circ R_1 w'$ such that $w' \models K_2 p$.

Nevertheless, there are other formulas where it remains true. For instance, we have $\models_{L^{*1D}} K_1 K_2 K_3 p \wedge K_2 K_1 K_3 p \rightarrow K_1 K_2 K_1 \dots K_2 \dots K_3 p$.

Question 1: What about $K_1 K_2 \varphi \wedge K_2 K_1 \varphi \rightarrow K_1 K_2 K_1 \varphi$ if φ do not contain agent 1 or 2? Do we have a characterisation or exhibit an interesting set of formulas φ such that $K_1 K_2 \varphi \wedge K_2 K_1 \varphi \rightarrow K_1 K_2 K_1 \varphi$ holds?

D. A compact representation

Last but not the least, you can remark that if we want to deal with model-checking, satisfiability problem and other algorithmic problems, we need a compact representation that an algorithm can manipulate. Worlds are difficult to manipulate: in particular, it is unadapted that $R_a(w)$ is infinite given a agent a and a world w . According to the Definition 8, the set W is infinite. Nevertheless, the semantics do not depend on positions of lamps and agents but only on how

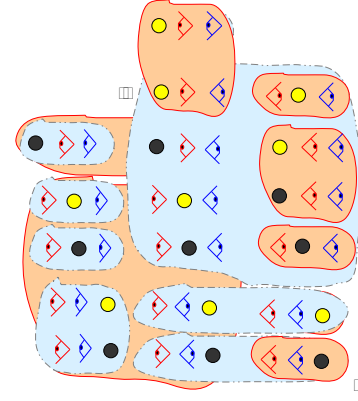


Fig. 2. Some worlds of the model \mathcal{M}

they are ordered on the line. For instance, $\begin{array}{|c|} \hline \begin{array}{c} \odot \quad \blacklozenge \\ p \quad a \end{array} \\ \hline \end{array}$ and

$\begin{array}{|c|} \hline \begin{array}{c} \odot \quad \blacklozenge \\ p \quad a \end{array} \\ \hline \end{array}$ stands for the same world. We can

define the notion of *description of a world* w : it is simply a total preorder over all propositions and agents appearing in a formula, plus d_{AGT} and π . Notice that we can do this because the space is a line. If our space were \mathbb{R}^n ($n \geq 2$), the notion of total preorder would unfortunately not be suited anymore.

Definition 7 (description of a world):

A *description of a world* w is a tuple $\langle \leq, d_{AGT}, \pi \rangle$ where:

- \leq is a total preorder over $AGT \cup ATM$;
- $d_{AGT} : AGT \rightarrow \{-1, +1\}$;
- $\pi : ATM \rightarrow \{\perp, \top\}$.

We can also define the epistemic relation between two description of a world w :

Definition 8 (epistemic relation):

Let $a \in AGT$. We define the *epistemic relation* R_a on the set of descriptions of worlds by $w R_a v$ iff:

- if $d_{AGT}(a) = +1$,
 - for all $x \in AGT \cup ATM$, $(x \leq_w a \text{ iff } x \leq_v a)$;
 - for all $x, y \in AGT \cup ATM$ such that $a \leq_w x$ and $a \leq_w y$, we have $(x \leq_w y \text{ iff } x \leq_v y)$;
 - for all $x \in AGT$, $a \leq_w x$ implies $d_{AGT_w}(x) = d_{AGT_v}(y)$;
 - for all $x \in ATM$, $a \leq_w x$ implies $\pi_w(x) = \pi_v(y)$.
- if $d_{AGT}(a) = -1$,
 - for all $x \in AGT \cup ATM$, $(x \geq_w a \text{ iff } x \geq_v a)$;
 - for all $x, y \in AGT \cup ATM$ such that $a \geq_w x$ and $a \geq_w y$, we have $(x \geq_w y \text{ iff } x \geq_v y)$;
 - for all $x \in AGT$, $a \geq_w x$ implies $d_{AGT_w}(x) = d_{AGT_v}(y)$;
 - for all $x \in ATM$, $a \geq_w x$ implies $\pi_w(x) = \pi_v(y)$.

In the same way, we can define an epistemic model. We can define truth conditions of a formula φ in \mathcal{L}_{AGT} over the set of descriptions of worlds, using the epistemic relation. We can prove that we obtain the same validities.

Definition 9 (extracting description of world from a world):

Given a world w , we define the description of world $d(w)$ by:

- for all $x, y \in AGT \cup ATM$, $x \leq_{d(w)} y$ iff $p(x) \leq_{\mathbb{R}} p(y)$ where $p(x)$ stands for $p_{AGT}(x)$ if $x \in AGT$ or $p_{ATM}(x)$ if $x \in ATM$;
- $d_{AGTw} = d_{AGTd(w)}$;
- $\pi_w = \pi_v$.

Proposition 4: For all $w \in W$, for all $\varphi \in \mathcal{L}_{AGT}$, $w \models \varphi$ iff $d(w) \models \varphi$.

Proof: By induction on φ . ■

In the case of one dimension, we simply rewrite mapping from ATM or AGT to real numbers into a total preorder over $ATM \cup AGT$. In the case of two or more dimensions, it is an open problem how to represent a world in a compact way.

III. MODEL-CHECKING AND SATISFIABILITY PROBLEM

For definitions for complexity class and for more details about the problem QSAT (quantified boolean formulas satisfiability problem), the reader may refer to [9].

A. Definitions

Now we are going to recall the classical problem of model-checking and satisfiability. The problem of model-checking consists on testing if a given formula φ is true in a given world w . Satisfiability problem consists to test if there exists a world w in which a given formula φ is true.

*Definition 10 (model-checking of $L^{*1D}_{AGT,ATM}$):*

Let AGT be a set of agents and ATM a set of atoms. We call *model-checking of $L^{*1D}_{AGT,ATM}$* problem the following problem:

- Input: a formula $\varphi \in \mathcal{L}_{AGT}$, a description of a world w where only atoms and agents occurring in φ are given;
- Output: Yes iff we have $w \models_{L^{*1D}} \varphi$. No, otherwise.

In the previous Definition, we give a description of a world w that is to say a total preorder over all agents and propositions occurring in φ where we say for each agent if he is look on the left or on the right and for each proposition if it is true or not. We do not care about propositions or agents not in the formula φ . The description of w is then *finite*.

*Definition 11 (L^{*1D}_{AGT} -satisfiability problem):*

Let AGT be a set of agents. We call *L^{*1D}_{AGT} -satisfiability problem* the following problem:

- Input: a formula $\varphi \in \mathcal{L}_{AGT}$;
- Output: Yes iff there exists a world w such that $w \models_{L^{*1D}} \varphi$. No, otherwise.

```

function check( $w, \varphi$ )
  match ( $\varphi$ )
     $\top$ :
      | return  $\top$ ;
     $p \in ATM$ :
      | return  $\top$  if  $p$  is true in  $w$ ;
      | return  $\perp$  if  $p$  is false in  $w$ ;
     $\psi_1 \wedge \psi_2$ :
      | return check( $w, \psi_1$ )  $\wedge$  check( $w, \psi_2$ );
     $\neg\psi$ :
      | return  $\neg$ check( $w, \psi$ );
     $K_a\psi$ :
      | for  $u \in R_a(w)$  do
          | if check( $u, \psi$ ) =  $\perp$  then
              | return  $\perp$ ;
          | endIf
      | endFor
      | return  $\top$ ;
  endMatch
endFunction
    
```

Fig. 3. A PSPACE-algorithm for model-checking of L^{*1D}_{AGT}

B. PSPACE-ness upper-bound of the two problems

In this subsection, we are going to give PSPACE-ness upper-bound of the model checking problem and also of the satisfiability problem. As you will see, the proof are directly given with algorithms using a polynomial amount of memory (Figures 3 and 4).

Proposition 5: Let AGT be any set of agents. The model-checking of L^{*1D}_{AGT} problem is in PSPACE.

Proof:

You can take a look at the recursive algorithm of Figure 3. We have to prove three points: terminaison, correctness and PSPACE-ness.

- 1) First let us prove terminaison by induction on φ . Let $\mathcal{T}(\varphi)$ be the property “for every world w , the call $check(w, \varphi)$ terminates”.
 - $check(w, \top)$ and $check(w, p)$ terminates. So $\mathcal{T}(\top)$ and $\mathcal{T}(p)$;
 - Let us prove that $check(w, K_a\psi)$ terminates. By induction, $\mathcal{T}(\psi)$ so every call $check(u, \psi)$ terminates. So the call $check(w, K_a\psi)$ terminates and $\mathcal{T}(K_a\psi)$;
 - Other cases are treated in the same manner.
- 2) Secondly, we have to prove correctness. Correctness corresponds to the property $\mathcal{C}(\varphi)$ defined by “for all world w , $w \models \varphi$ iff $check(w, \varphi) = \top$ ”. We also prove $\mathcal{C}(\varphi)$ for all formula φ by induction.
- 3) Finally, we prove that $check$ only requires a polynomial amount of memory. Just be careful at the line “**for** $u \in R_a(w)$ **do**”: although $R_a(w)$ may be of size exponential we do not compute it. Here we only enumerate here elements of $R_a(w)$ one by one. This can be done using only a linear amount of memory. This part is technical but I will nevertheless give some details how to implement an enumeration of elements of $R_a(w)$.
The block:

```

for  $u \in R_a(w)$  do
  if  $check(u, \psi) = \perp$  then
    return  $\perp$ ;
  endIf
endFor
    
```

can be rewritten in a unreadable block using a linear amount of memory in (*):

```

 $u := first\_permutation(w)$ 
while  $\neg is\_last\_permutation(u)$  do
  if  $u \in R_a(w)$ 
    if  $check(u, \psi) = \perp$  then
      return  $\perp$ ;
    endIf
  endIf
   $u := next\_permutation(u)$ ;
endWhile
    
```

where:

- assuming we have an order $<$ over permutations of elements appearing in w , $first_permutation(w)$ gives, using a linear amount of memory, the first permutation we can make with elements of w ; For

instance, if $w = \begin{smallmatrix} \diamond \\ 0 \end{smallmatrix} \begin{smallmatrix} \star \\ p \end{smallmatrix}$, $first_permutation(w)$

can be $\begin{smallmatrix} \diamond \\ 0 \end{smallmatrix} \blacksquare$;

- $next_permutation(u)$ is a function, using a linear amount of memory, giving the $<$ -successor of u ; For instance, we may have:

– $next_permutation(\begin{smallmatrix} \diamond \\ 0 \end{smallmatrix} \blacksquare) = \begin{smallmatrix} \diamond \\ 0 \end{smallmatrix} \star$;

– $next_permutation(\begin{smallmatrix} \diamond \\ 0 \end{smallmatrix} \star) = \begin{smallmatrix} \diamond \\ 0 \end{smallmatrix} \star$;

– $next_permutation(\begin{smallmatrix} \diamond \\ 0 \end{smallmatrix} \star) = \begin{smallmatrix} \blacksquare \\ p \end{smallmatrix} \begin{smallmatrix} \diamond \\ 0 \end{smallmatrix}$ etc.

- $is_last_permutation(u) = \top$ iff u has no $<$ -successor.

Now, we can prove by induction on φ the following property for all φ , $\mathcal{P}(\varphi)$ defined as “for all world w , the call $check(w, \varphi)$ needs $O(|\varphi| \times |w|)$ memory cells”.

- $\mathcal{P}(\top)$ and $\mathcal{P}(p)$ are true;
- Let us prove $\mathcal{P}(\psi_1 \wedge \psi_2)$. The first call $check(w, \varphi_1)$ needs $O(|\varphi_1| \times |w|)$ by hypothesis of induction. Then we can release all the memory cells used for the sub-call $check(w, \varphi_1)$ and we can treat the call $check(w, \varphi_2)$. It needs $O(|\varphi_2| \times |w|)$. Hence, the sub-call $check(w, \varphi_1 \wedge \varphi_2)$ needs $\max(O(|\varphi_1| \times |w|), O(|\varphi_2| \times |w|)) = O(|\varphi| \times |w|)$. So $\mathcal{P}(\psi_1 \wedge \psi_2)$.

```

function  $sat(\varphi)$ 
   $w := choose\_world\_with\_symbols\_in(\varphi)$ 
  return  $check1(w, \varphi)$ 
endFunction
    
```

Fig. 4. A PSPACE-algorithm for satisfiability problem of $L^{\exists 1D}_{AGT}$

$card(AGT)$	$L^{\exists 1D}_{AGT} - md$	$L^{\exists 1D}_{AGT} - sat$
1	Σ_1 -hard, in Δ_2 -hard	Σ_2 -complete
$n \in \mathbb{N}, n \geq 2$	Σ_n -hard, in ??	Σ_{n+1} -hard, in ??
∞	PSPACE-complete	PSPACE-complete

$card(AGT)$	$S5_{card(AGT)} - sat$
1	NP-complete
$n \geq 2$	PSPACE-complete
∞	PSPACE-complete

Fig. 5. Table of complexities

- Now, we prove $\mathcal{P}(K_a\psi)$. By induction, every sub-call $check(w, \psi)$ needs at most $O(|\psi| \times |w|)$ memory cells. Furthermore, we need $O(|w|)$ for $first_permutation(w)$, $is_last_permutation(u)$ and $next_permutation(u)$ and also to keep the local variable u in memory. So we need, $O(|\psi| \times |w|) + O(|w|) = O(|\varphi| \times |w|)$.

Finally, $\mathcal{P}(\varphi)$ is true for all φ . In other words, the algorithm of Figure 3 only use a polynomial number of memory cells (we take in account (*)).

Proposition 6: Let AGT be any set of agents. The $L^{\exists 1D}_{AGT}$ -satisfiability problem is in PSPACE.

Proof: You can read the algorithm of Figure 4. The algorithm consists in guessing non-deterministically a world w and then call the routine $check$ of Figure 3 to check if φ is true in w . So, the problem is NPSpace, hence from Savitch's theorem [12], it is PSPACE.

Now we are going to investigate more in details complexities of the model checking and satisfiability problem depending on the size of AGT . The table of Figure 5 sums up all results we have. There is also the recall of complexity results about $S5_n$ satisfiability problem as comparison.

C. When AGT is infinite: PSPACE-complete

We recall the complexity result about QBF formulas satisfiability problem:

Theorem 1: The QSAT-problem defined as following:

- Input: a formula $\varphi = \exists \vec{p}_1 \forall \vec{p}_2 \exists \vec{p}_3 \forall \vec{p}_4 \dots Q_n \vec{p}_n \psi$ where:
 - n is any integer;
 - ψ is a boolean formula;
 - and $Q_i = \forall$ if i is even and $Q_i = \exists$ if i is odd;
 - \vec{p}_j is a finite set of variables for each j .
- Output: Yes iff $\models_{QBF} \varphi$. No, otherwise.

is PSPACE-complete.

Now the following Proposition gives a translation of a QBF-instance into a $L^{\exists 1D}$ -model-checking instance or a $L^{\exists 1D}$ -satisfiability problem instance.

Proposition 7: Let $\varphi = \exists \vec{p}_1 \forall \vec{p}_2 \exists \vec{p}_3 \forall \vec{p}_4 \dots Q_n \vec{p}_n \psi$ be a formula of the logic QBF. We define $f(\varphi)$ by induction:

- $f(\psi) = \psi$;
- $f(\forall \vec{p}_i \dots Q_n \vec{p}_n \psi) = K_{i-1}(put_i \rightarrow f(\exists \vec{p}_{i+1} \dots Q_n \vec{p}_n \psi))$;
- $f(\exists \vec{p}_i \dots Q_n \vec{p}_n \psi) = \hat{K}_{i-1}(put_i \wedge f(\forall \vec{p}_{i+1} \dots Q_n \vec{p}_n \psi))$;

where:

- $put_a = \bigwedge_{i \in \{a+1, \dots, 2n\}} \neg K_a^{if} \vec{p}_i \wedge \bigwedge_{i \in \{1, \dots, a\}} K_a^{if} \vec{p}_i$;
- $K_a^{if} \vec{p} = \bigwedge_{q \in \vec{p}} K_a^{if} q$;
- $K_a^{if} q = K_a q \vee K_a \neg q$.

We have equivalence between:

- $\models_{QBF} \varphi$;
- $put_1 \wedge f(\forall \vec{p}_2 \exists \vec{p}_3 \forall \vec{p}_4 \dots Q_n \vec{p}_n \psi)$ is L^{*1D}_{AGT} -satisfiable;
- and $w \models_{L^{*1D}} f(\varphi)$ where $w \in W_0$ where W_0 is the set of all worlds where agent 0 is completely on the left

looking to the left. (we note $W_0 = \langle \begin{smallmatrix} \diamond \\ 0 \end{smallmatrix} \dots \rangle$).

Proof: We are going to note for all $U \subseteq W$, $U \models \varphi$ iff for all $u \in U$, $u \models \varphi$. We are going to prove by induction $\models_{QBF} \varphi$ iff $W_0 \models_{L^{*1D}} f(\varphi)$. We are going to note for all $i \in \mathbb{N}$, for all valuation $\nu[\vec{p}_1, \dots, \vec{p}_i]$,

$$W_i(\nu[\vec{p}_1, \dots, \vec{p}_i]) \stackrel{\parallel def}{=} \langle \begin{smallmatrix} \diamond \\ 0 \end{smallmatrix} \dots \rangle$$

$$\langle \begin{smallmatrix} \diamond \\ 0 \end{smallmatrix} \dots \rangle \quad \langle \begin{smallmatrix} \odot \\ \vec{p}(p_1) \end{smallmatrix} \dots \rangle \quad \langle \begin{smallmatrix} \diamond \\ 1 \end{smallmatrix} \dots \rangle \quad \langle \begin{smallmatrix} \odot \\ \nu(\vec{p}_2) \end{smallmatrix} \dots \rangle \quad \langle \begin{smallmatrix} \diamond \\ 2 \end{smallmatrix} \dots \rangle \quad \dots \quad \langle \begin{smallmatrix} \odot \\ \nu(\vec{p}_i) \end{smallmatrix} \dots \rangle \quad \langle \begin{smallmatrix} \diamond \\ i \end{smallmatrix} \dots \rangle \dots$$

The induction hypothesis is:

$$\nu[\vec{p}_1, \dots, \vec{p}_{i-1}] \models_{QBF} Q_i \vec{p}_i \dots Q_n \vec{p}_n \psi \text{ iff}$$

$$W_{i-1}(\vec{p}_1, \dots, \vec{p}_{i-1}) \models_{L^{*1D}} f(Q_i \vec{p}_i \dots Q_n \vec{p}_n \psi)$$

The basis case correspond to $i = n+1$. It is the propositional case. We have:

$$\nu[\vec{p}_1, \dots, \vec{p}_n] \models_{QBF} \psi \text{ iff } W_n(\vec{p}_1, \dots, \vec{p}_n) \models_{L^{*1D}} \psi.$$

Now we can attack the induction case. Let us prove for i odd. $\nu[\vec{p}_1, \dots, \vec{p}_{i-1}] \models_{QBF} Q_i \vec{p}_i \dots Q_n \vec{p}_n \psi$ means that there exists a valuation $\nu(p_i)$ such that $\nu[\vec{p}_1, \dots, \vec{p}_i] \models_{QBF} Q_{i+1} \vec{p}_{i+1} \dots Q_n \vec{p}_n \psi$. By induction, it means that $W_i(\vec{p}_1, \dots, \vec{p}_i) \models_{L^{*1D}} f(Q_{i+1} \vec{p}_{i+1} \dots Q_n \vec{p}_n \psi)$.

But for all $w_{i-1} \in W_{i-1}(\vec{p}_1, \dots, \vec{p}_{i-1})$ and for all $w_i \in W_i(\vec{p}_1, \dots, \vec{p}_i)$, we have:

- $w_{i-1} R_{i-1} w_i$;
- $w_i \models put_i$. Indeed, for all $j > i$, we have $w \models \neg K_i^{if} \vec{p}_j$ because agent j does not see lamps \vec{p}_j in w_i . On the contrary, for all $j < i$, we have $w \models K_i^{if} \vec{p}_j$ because agent i do see lamps \vec{p}_j in w_i (the valuation of lamps \vec{p}_j is the same in all worlds $u \in R_i(w_i)$). The technical proof of $w_i \models put_i$ is left to the reader.

As $f(\exists \vec{p}_i \dots Q_n \vec{p}_n \psi) = \hat{K}_{i-1}(put_i \wedge f(\forall \vec{p}_{i+1} \dots Q_n \vec{p}_n \psi))$, we have:

$W_{i-1}(\vec{p}_1, \dots, \vec{p}_{i-1}) \models_{L^{*1D}} f(\exists \vec{p}_i \dots Q_n \vec{p}_n \psi)$. We ensure that it is equivalent.

The case where i is even is similar. ■

Immediately from this translation, we deduce the lower bound for model-checking in L^{*1D} .

Corollary 2: Let AGT an infinite enumerable set of agents. The model-checking problem of L^{*1D}_{AGT} is PSPACE-hard.

Proof: Reduction via Proposition 7 and Theorem 1 in order to the PSPACE-hardness and Proposition 6. ■

In the same way we have:

Corollary 3: Let AGT an infinite enumerable set of agents. The satisfiability problem of L^{*1D}_{AGT} is PSPACE-hard.

D. When AGT is finite

We recall the complexity result about QBF formulas satisfiability problem but when the nesting of \forall and \exists is bounded by a fixed integer n .

Theorem 2: Let n be a integer. The QSAT $_n$ -problem defined as following:

- Input: a formula $\varphi = \exists \vec{p}_1 \forall \vec{p}_2 \exists \vec{p}_3 \forall \vec{p}_4 \dots Q_n \vec{p}_n \psi$ where ψ is a boolean formula, and $Q_i = \forall$ if i is even and $Q_i = \exists$ if i is odd;
- Output: Yes iff $\models_{QBF} \varphi$. No, otherwise.

is Σ_n -complete.

The Theorem 2 only differ from Theorem 1 by the fact that n is no more a input of the problem but is now fixed inside the problem. For each integer n , we have defined the QSAT $_n$ -problem. There is a enumerable number of problems.

In the same way, this precise complexity result of QBF combined with the translation of QBF to L^{*1D} allows us to have complexity lower bounds of model-checking and satisfiability problem when the cardinality of the set AGT is finite and fixed.

Corollary 4: Let AGT a finite set of agents. The model-checking problem of L^{*1D}_{AGT} is $\Sigma_{card(AGT)}$ P-hard.

Proof: Reduction via Proposition 7 and Theorem 2. ■

Corollary 5: Let AGT a finite set of agents. The satisfiability problem of L^{*1D}_{AGT} is $\Sigma_{card(AGT)+1}$ P-hard.

Proof: Reduction via Proposition 7 and Theorem 2. ■

E. When $card(AGT) = 1$

Unfortunately we do not have a precise complexity upper-bound for those problems in the general case when $card(AGT)$ is finite. Nevertheless, we have the exact complexity when $card(AGT) = 1$.

Proposition 8: The model-checking problem of $L^{*1D}_{\{1\}}$ is in Δ_2P .

Proof: The figure 6 gives us an Δ_2P -algorithm (a P-algorithm with NP-oracles) for the model-checking problem of $L^{*1D}_{\{1\}}$. Given a world w , first we compute the V of propositions occurring in φ that the agent 1 sees and I the set of propositions the agent 1 does not see. Then we can replace each occurrence p of a proposition p from V


```

function check1( $w, \varphi$ )
     $V :=$  set of variables that agent 1 sees in  $w$ ;
     $I :=$  set of variables that agent 1 does not see in  $w$ ;
     $\psi := \varphi$  in which we replace each  $p \in V$  by  $\pi_w(p)$ ;
     $\psi := \varphi$  in which we replace each  $p \in I$  not in the scope
    of a  $K_1$  by  $\pi_w(p)$ ;
    while there exists  $K_1\chi$  subformula of  $\psi$ , where  $\chi$  is a
    boolean formula do
        if oracle - sat( $\neg\chi$ ) then
            |  $\psi := \psi$  in which we replace  $K_1\chi$  by  $\perp$ ;
        else
            |  $\psi := \psi$  in which we replace  $K_1\chi$  by  $\top$ ;
        endIf
    endWhile
    return PCL( $\{\perp, \top\}$ ) - sat( $\psi$ );
endFunction
    
```

 Fig. 6. A Δ_2P -algorithm for model checking of $L^{\exists 1D}_{\{1\}}$

```

function sat( $\varphi$ )
     $w :=$  choose_world_with_symbols_in( $\varphi$ )
    return check1( $w, \varphi$ )
endFunction
    
```

 Fig. 7. Optimal Σ_2P -algorithm for satisfiability problem of $L^{\exists 1D}_{\{1\}}$

in φ by the corresponding valuation $\pi_w(p)$. Concerning a proposition $p \in I$, we only replace occurrences which are not in the scope of a K_1 . For instance, if $p \in I$, $q \in V$, and $\pi_w(p) = \top, \pi_w(q) = \perp$ $p \wedge q \vee K_1(p \vee q)$ is replaced by $\top \wedge \perp \vee K_1(p \vee \perp)$. Then we test satisfiability of boolean formulas $\neg\chi$ such that $K_1\chi$ is a subformula of ψ and replace $K_1\chi$ by \perp if $\neg\chi$ is satisfiable and by \top otherwise. At the end, we obtain a boolean formula ψ containing no variables. We test its satisfiability with $PCL(\{\perp, \top\}) - sat(\psi)$. Notice the **while**-loop is done in linear time because there are a linear number of subformulas in ψ . ■

Proposition 9: The satisfiability problem of $L^{\exists 1D}_{\{1\}}$ is Σ_2P -complete.

Proof:

The hardness comes from Corollary 5. The Figure 7 gives us an Σ_2P -algorithm (a NP-algorithm with NP-oracles) for the satisfiability problem of $L^{\exists 1D}_{\{1\}}$. ■

F. When AGT and ATM are both finite

Proposition 10: Let ATM a finite set of agents and $AGT = \{1\}$. The satisfiability problem and model-checking of $L^{\exists 1D}_{\{1\}}$ is in P.

Proof: We adapt algorithms of the figure 6 and 7 in order to have an optimal polynomial algorithm. More precisely:

- You replace *choose_world_with_symbols_in* in Figure 7 by a loop over all worlds. You can notice that the set of all possible worlds is fixed, that is to say it does not depend on φ ;
- *oracle - sat* can now run in polynomial time because there is a fixed number of propositions. ■

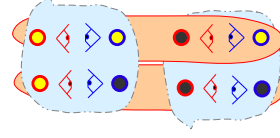


Fig. 8. World for Muddy children

IV. PUBLIC ANNOUNCEMENTS

As done in [11] we can extend our framework with public announcements. This is essentially motivated by modelling examples like Muddy children. With public announcements, an agent will be able to learn something about the part of the actual world which he can not see. The technique is classical: we add an operator $[\varphi!]$ and we define semantics as in $S5_n$.

A. Definitions

Our new language $\mathcal{L}_{AGT}^!$ is defined by the following BNF:

$$\varphi ::= p \mid \varphi \wedge \varphi \mid \neg\varphi \mid K_a\psi \mid [\varphi!]\varphi$$

where $p \in ATM$ and $a \in AGT$.

From now, we do not only parameter \models with a world but also with the set of worlds.

Definition 12 (truth conditions):

Let U a set of worlds ($U \subseteq W$). Let $w \in U$. We define $U, w \models \varphi$ by induction:

- $U, w \models p$ iff $\pi(p) = \top$
- $U, w \models \varphi \wedge \psi$ iff $U, w \models \varphi$ and $U, w \models \psi$;
- $U, w \models \neg\varphi$ iff $U, w \not\models \varphi$;
- $U, w \models K_a\psi$ iff for all $w' \in U$, $wR_a w'$ implies $U, w' \models \psi$;
- $U, w \models [\varphi!]\psi$ iff $U, w \models \varphi$ implies $U \cap \{w' \in U \mid U, w' \models \varphi\}, w \models \psi$.

The set of validities we obtain is noted $L^{\exists 1D}! = \{\varphi \mid W, w \models \varphi\}$ where W is the set of all worlds defined Definition 8.

B. Example

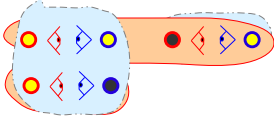
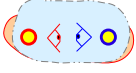
Now we are going to study the Muddy children example. This example is also studied in [11]. You can also find this example in [4] and [3] with more than two children. The situation is the following: there are two children named 1 and 2. Their foreheads are dirty. They see each others. The situation is represented by the world w shown in Figure 8 in the top left. One child do not know if he is dirty or not but he knows the state of the forehead of the other one. We introduce two propositions: p stands for “1’s forehead is dirty” and q stands for “1’s forehead is dirty”.

We have:

- $W, w \models K_1q \wedge K_2p$;
- $W, w \models \neg K_1p \wedge \neg K_1(\neg p) \wedge \neg K_2(\neg q)$.

Then:

- the father says at least one of them are dirty;


 Fig. 9. World for Muddy children after having announced φ_1

 Fig. 10. World for Muddy children after having announced φ_1 and φ_2

- the children answer that they do not know whether they are dirty or not.

Formally, we also have:

$$W, w \models [\varphi_1!][\varphi_2!](K_1p \wedge K_2q)$$

where:

- $\varphi_1 = p \vee q$;
- $\varphi_2 = (\neg K_1p \vee \neg K_1\neg p) \wedge (\neg K_2q \vee \neg K_2\neg q)$.

We verify that after having announced φ_1 , we only consider worlds in Figure 9. Then we only consider the initial world w drawn in 10.

C. Complexity

Because $L^{\ast 1D^!}$ is a conservative extension of $L^{\ast 1D}$, we inherit from the lower bound results both for model-checking and satisfiability. In fact, we keep the PSPACE-ness upper-bound with public announcements.

Proposition 11: The model-checking and satisfiability problem in $L^{\ast 1D^!}$ is PSPACE-complete.

Proof: The Figure 11 gives an algorithm for model-checking. As usual, w is a world, φ is a formula. The second argument C is a list of formulas and stands for the *context*: if $C = []$ (empty list), it corresponds to the whole set of worlds otherwise it is a list of announced formulas used to update the model. More precisely, let us define:

- $W^[] = W$;
- $W^{[\psi_1:C']} = \{w \in W^{C'} \mid W^{C'}, w \models \psi_1\}$.

We want:

- $mc(w, C, \varphi)$ returns true iff $W^C, w \models \varphi$;
- $inupdatedM(w, C)$ returns true iff $w \in W^C$.

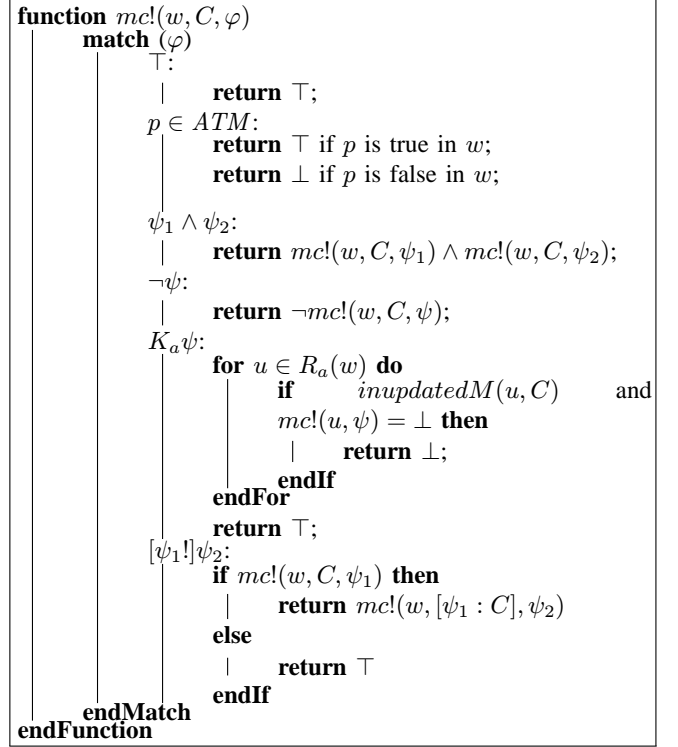
We have to prove termination, correctness and complexity. Let us begin to prove termination. First of all, we are going to introduce an order \prec over all possible inputs (C, φ) of the function $mc!$ of Figure 11.

We define $(C, \varphi) \prec (D, \psi)$ by:

- $|C| + |\varphi| < |D| + |\psi|$;

- or $\begin{cases} |C| + |\varphi| = |D| + |\psi| \\ \text{and } |\varphi| < |\psi| \end{cases}$

where


 Fig. 11. Algorithm for model-checking in $L^{\ast 1D^!}$

- $|\varphi|, |\psi|$ denotes the length (number of symbols) in φ, ψ ;
 - $|C|, |D|$ denotes also the number of symbols in C, D .
- More precisely:

- $|| = 0$;
- $||[\psi_1:C']| = |\psi_1| + |C'|$.

The order \prec is well-founded and we can use it to prove termination by induction.

- Basic case: (\emptyset, p) etc.
- Induction case: you just have to see that $mc!(w, C, \varphi)$ will only call $mc!(u, D, \psi)$ with $(D, \psi) \prec (C, \varphi)$. For instance, when $mc!(w, C, \varphi)$ calls $inupdatedM(u, C)$ which calls $mc!(w, C', \psi)$, we have $|C'| + |\psi| = |C| < |C| + |\varphi|$.

Correction and the fact that the algorithm runs using a polynomial space can also be proved by induction using the order \prec .

The hardness comes from the fact that $L^{\ast 1D^!}$ is a conservative extension of $L^{\ast 1D}$ and the model checking of $L^{\ast 1D}$ is PSPACE-hard (Corollary 2).

Concerning the satisfiability, we can make the same remark than in the proof of Proposition 6. ■

The upper-bound in special cases (*AGT* finite etc.) has not been studied yet.

From now, we are to discuss about the implementation and develop the example of the Muddy children.

```

function inputatedM(w, C)
  match (C)
    []: return ⊤;
    [ψ : C']: mc!(w, C', ψ);
  endMatch
endFunction

```

Fig. 12. Algorithm for testing if a world w is in the updated model formulas in C

V. IMPLEMENTATION

You can find an implementation on the Web site. You can put positions and directions of agents and positions and states of lamps on your own. Then you can write down a formula and check if your formula is true in the world you have drawn.

This program offers a concrete example to illustrate epistemic logic to students.

A. Description

The program is written in Scheme for the easy use of data structures and recursive programming. Haskell could also be a well-suited language especially for the lazy evaluation enabling us to write a program which seems to use an exponential amount of memory whereas it uses only a polynomial amount of memory. Here are the main Scheme functions:

- (mc world formula) computes if the formula formula is true in the world world;
- (mc-with-context world context formula) computes if the formula formula is true in the world world but we restrict our check computations only on worlds satisfying the formula context;
- (worldset-delete-not-satisfying worldset formula) removes from the set of world worldset all worlds which does not satisfy the formula formula. This function is used to deal with updated models;
- (world-getpossibleworlds world agent) computes the set of all possible worlds for agent agent in world world.

In order to be human readable, the implementation does not run in polynomial space but in exponential time. For instance the function (world-getpossibleworlds w a) computes really *all* worlds in $R_a(w)$.

B. Practising Muddy children

You can describe the *current situation* (world w on the top left in Figure 8 by $((p \#t) (1 <) (2 >) (q \#t))$. Notice that we are not going to construct the Kripke structure by hand. When you draw a Kripke model, you can easily mistakes all the more so the model is theoretical. Here we just enjoy specifying *graphically* the situation. The Kripke structure is then generated on-the-fly by the algorithm. You can test if $W, w \models K_1p \wedge K_2q$ by calling

```
(mc '((p #t) (1 <) (2 >) (q #t)) ((1 knows p) and (2 knows q))).
```

The function returns #f meaning that we do not have $W, w \models K_1p \wedge K_2q$.

We ask the computer the different worlds the agent 1 imagine. To do this we write

```
(world-getpossibleworlds '((p #t) (1 <) (2 >) (q #t)) 1).
```

The system gives:

```
((p #t) (1 <) (2 >) (q #t))
((p #f) (1 <) (2 >) (q #t))).
```

We can now test if the formula $W, w \models [\varphi_1!][\varphi_2!](K_1p \wedge K_2q)$. You simply write

```
(mc '((p #t) (1 <) (2 >) (q #t))
(announce (p or q) (announce ((not (1 knows p)) and (not (2 knows q)))) ((1 knows p) and (2 knows q)))).
```

The system answers #t.

VI. CONCLUSION

The epistemic logic $S5_n$ is a general and theoretical framework for the representation of knowledge. In this paper, we have studied a spatially grounded epistemic logic. We have investigated two aspects of knowledge learning:

- With L^{*1D} , we can reason about what agents know by learning only with their eyes (when they are located on a line space);
- With $L^{*1D!}$ we can reason about what agents know by looking at their environments and by listening to public announcements.

Of course the case of the line is restrictive. The case of the plane or of the space may be more interesting. Nevertheless, this paper gives complexity results for model-checking and satisfiability problem for the case of the line. Even the line looks like easy, problems are already PSPACE-complete if the number of agents is not bounded. We conjecture that the complexity of this logic for dimension $n \geq 2$ remains PSPACE-complete.

From now, there are two main perspectives: to adapt this logic to the case of two dimensions [1] and to study properly complexity of model checking and satisfiability with/without public announcements. Other perspectives are numerous:

- fill the Figure 5. The exact complexity classes of model checking and satisfiability L^{*1D}_{AGT} when AGT is finite are still open questions;
- Study and implement the logic with agents and lamps in the plane [1] and compare it to the logic in the line. Writing down the semantics is quite easy: you just have to replace \mathbb{R} by \mathbb{R}^2 in Definition 8 and tune the definition of directions and Definition 3. The main difficulty is to find a compact representation in order to deal with the model checking and satisfiability problem. In two dimensions it is no more possible to consider a total preorder on elements. Finding a good equivalent of Definition 7 satisfying Proposition 4 in the case of dimension 2 or more is still an open problem.
- Study the logic in the 3D-space and compare it to the one in the plane (I guess we obtain the same validities);

- Find an axiomatization of those logics in order to understand better how they work;
- Study if it is possible to have a normal form (like for S5, where all formulas are equivalent to a formula of modal degree 1 [6]);
- Extend with a common knowledge operator. Will the complexity of the satisfiability problem also increase and become EXPTIME-complete?
- Extend with private communications between agents.

Acknowledgements.

I thank Philippe Balbiani, Olivier Gasquet, Andreas Herzig, Emiliano Lorini and the reviewers for their different helpful remarks.

REFERENCES

- [1] Edwin Abbott Abbott. *Flatland*. Basil Blackwell, 1884.
- [2] Jon Barwise and John Etchemendy. *Tarski's World: Version 4.0 for Macintosh (Center for the Study of Language and Information - Lecture Notes)*. Center for the Study of Language and Information/SRI, 1993.
- [3] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995.
- [4] V. Goranko and M. Otto. *Handbook of Modal Logic*, chapter Model Theory of Modal Logic, pages 255–325. Elsevier, 2006.
- [5] Joseph Y. Halpern and Yoram Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54(3):319–379, 1992.
- [6] G. E. Hughes and M. J. Cresswell. *An Introduction to Modal Logic*. Methuen and Co., 1968.
- [7] Ethan Kennerly, Andreas Witzel, and Jonathan A. Zvesper. Thief belief (extended abstract). Presented at Logic and the Simulation of Interaction and Reasoning 2 (LSIR2) Workshop at IJCAI-09, July 2009.
- [8] E. Lorini, L. Tummolini, and A. Herzig. Establishing mutual beliefs by joint attention: towards a formal model of public events. In Monica Bucciarelli Bruno G. Bara, Lawrence Barsalou, editor, *Proceedings of the XXVII Annual Conference of the Cognitive Science Society*, pages 1325–1330. Lawrence Erlbaum, 2005.
- [9] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- [10] Rohit Parikh, Lawrence S. Moss, and Chris Steinsvold. Topology and epistemic logic. In *Handbook of Spatial Logics*, pages 299–341. 2007.
- [11] Jan Plaza. Logics of public communications. *Synthese*, 158(2):165–179, 2007.
- [12] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.
- [13] Hans P. van Ditmarsch, Wiebe van der Hoek, Ron van der Meyden, and Ji Ruan. Model checking russian cards. *Electr. Notes Theor. Comput. Sci.*, 149(2):105–123, 2006.

Games for Learning

A Sabotage Approach

Nina Gierasimczuk, Lena Kurzen and Fernando R. Velázquez-Quesada
 Institute for Logic, Language and Computation
 Universiteit van Amsterdam
 Email: {N.Gierasimczuk | L.M.Kurzen | F.R.VelazquezQuesada}@uva.nl

Abstract—In formal approaches to inductive learning, the ability to learn is understood as the ability to single out a correct hypothesis from a range of possibilities. Although most of the existing research focuses on the characteristics of the learner, in many paradigms the significance of the teacher’s abilities and strategies is in fact undeniable. Motivated by this observation, in this paper we highlight the interactive nature of learning by proposing a game-theoretical and logical approach. We consider learning as a *sabotage-type* game between *Teacher* and *Learner*, and present different variants based on the level of cooperativeness and the actions available to the players. We characterize the existence of a winning strategy in such games by formulas of *Sabotage Modal Logic*, analyzing also their complexity. Our work constitutes the first step towards a unified game-theoretical and logical approach to formal learning theory.

I. INTRODUCTION

The objective of this paper is to investigate how logics for interaction in multi-agent systems can be used to reason about strategic abilities and information flow during the *learning* process. Formal learning theory (see e.g. [4]) is concerned with the process of inductive inference: it formalizes the process of inferring general conclusions from partial, consecutively given information, as in the case of language learning (inferring grammars from sentences) and scientific inquiry (drawing general conclusions from partial experiments). We can think of this general process as a game between two players: *Learner* and *Teacher*. The game starts with a class of possible worlds from which *Teacher* chooses the actual one, and *Learner* has to find out which one it is. *Teacher* provides information about the world in an inductive manner, and whenever *Learner* receives a piece of information, he picks a conjecture from the initial class, indicating which one he thinks is the case. Several conditions can be defined for the success of the learning process: we can require that *Learner* arrives at a correct hypothesis (finite identification), or that the sequence of *Learner*’s conjectures converges to a correct hypothesis (identification in the limit) [3].

We give a high-level analysis of the described process. First, we treat learning as a procedure of singling out one correct hypothesis from a range of possibilities. Second, we see this procedure not as a one-move choice; instead, we allow many steps of update before the conclusion is

reached. These two properties make our notion of learning different from the concept of learning formalized as epistemic update in *Dynamic Epistemic Logic* (see e.g. [2]), where the word “learning” is often used as a synonym of “getting to know” and is usually represented as a one-step epistemic update. Moreover, in our approach we pay attention to the strategies for teaching, highlighting the fact that restricted power and knowledge of the learner can be compensated by additional insights and intentions of the teacher.

The paper is structured as follows. Section II introduces the framework of learning as Sabotage Games, shows how sabotage modal logic can express the existence of winning strategies in three different versions of Sabotage Learning Games and gives complexity results for them. Section III analyzes Sabotage Learning Games in which the players do not need to move in alternation. Section IV presents a refined interactive view on teaching based on existing learning algorithms. Section V concludes.

II. LEARNING AS A SABOTAGE GAME

Our work is motivated by the *learning from queries and counterexamples* model [1]. In that paradigm, the goal of *Learner* is to recognize an initially unknown language \mathcal{L} . In order to do this, he is allowed to ask *Teacher* two types of questions: about the membership of a certain string to \mathcal{L} , and about the equivalence of his conjecture (another language) to \mathcal{L} . When answering those questions, *Teacher* does not have any freedom — her responses are restricted by \mathcal{L} . However, a negative answer to the second question is accompanied by a counterexample, which plays the role of a hint for *Learner*. This is the only point of the procedure in which *Teacher* has a relative freedom of choice, and in fact, the informativeness of the string given as a counterexample influences the effectiveness of the learning process. We want to focus on this aspect of learning and show that the “profile” of *Teacher* is relevant for the learning process. We consider several possible scenarios — we describe games in which *Teacher* is either helpful or unhelpful, and *Learner* is either eager or unwilling to learn.

Let us consider a very simple “classroom” situation with one teacher and one learner. From our high-level

TABLE I
CORRESPONDENCE WITH LEARNING MODEL

Learning Model	Sabotage Games
hypotheses	states
correct hypothesis	goal state
possibility of a mind change from hypothesis a to hypothesis b	edge from state a to b
a mind change from hypothesis a to hypothesis b	transition from state a to b
giving a counterexample that eliminates the possibility of a mind change from a to b	removing a transition between a and b

perspective, learning is a step-by-step process through which Learner changes his information state, and the process is successful if he eventually reaches a state representing the goal. The information Teacher provides can be seen as feedback about Learner's current conjecture, allowing him to rule out possible changes of mind. We can represent the situation as a graph whose vertices represent Learner's possible information states and edges stand for transitions between them. During the learning process, Learner can change his information state by moving along the edges and Teacher can cut off edges, thereby preventing Learner from making certain transitions. One state is associated with the learning goal: if Learner reaches it, we say that the learning process has been successful. The correspondence between the learning model from formal learning theory and our proposal is described in Table I.

Observe that in learning from queries and counterexamples, a counterexample results in the absolute removal of some initially possible hypothesis. Our setting generalizes this idea: the removal of a transition need not make the target vertex unreachable.

A. Sabotage Games

Our perspective on learning leads naturally to the framework of Sabotage Games [6], [11]. A sabotage game is played in a directed multi-graph, with two players, *Runner* and *Blocker*, moving in alternation with *Runner* being the first. *Runner* moves by making a single transition from the current vertex; *Blocker* moves by deleting a *single* edge from *any* part of the graph. We begin by defining the structure in which a Sabotage Game takes place.

Definition 2.1 ([6]): A *directed multi-graph* is a tuple $G = (V, E)$ where V is a set of vertices and $E : V \times V \rightarrow \mathbb{N}$ is a function indicating the number of edges between any two vertices.

The Sabotage Game is defined as follows.

Definition 2.2 ([6]): A *Sabotage Game* $SG = \langle V, E, v, v_g \rangle$ is given by a directed multi-graph (V, E) and two vertices $v, v_g \in V$. Vertex v represents the position of *Runner* and v_g represents the goal state.

Each match is played as follows: the initial position $\langle E_0, v_0 \rangle$ is given by $\langle E, v \rangle$. Round $k + 1$ from position $\langle E_k, v_k \rangle$ consists of *Runner* moving to some v_{k+1} such that $E(v_k, v_{k+1}) > 0$, and then *Blocker* removing an edge (v, v') such that $E_k(v, v') > 0$. The new position is $\langle E_{k+1}, v_{k+1} \rangle$, where $E_{k+1}(v, v') := E_k(v, v') - 1$ and, for every $(u, u') \neq (v, v')$, $E_{k+1}(u, u') := E_k(u, u')$. The match ends if a player cannot make a move or if *Learner* reaches the goal state, which is the only case in which he wins.

Remark 1: It is easy to see that Sabotage Games have the *history-free determinacy property*: if one of the players has a winning strategy then she has a winning strategy that depends only on the current position. Then, each round can be viewed as a transition from a Sabotage Game $SG = \langle V, E_k, v_k, v_g \rangle$ to another Sabotage Game $SG' = \langle V, E_{k+1}, v_{k+1}, v_g \rangle$, since previous moves become irrelevant. We will use this fact through the whole paper. Also, by edges and vertices of $SG = \langle V, E, v, v_g \rangle$, we will mean edges and vertices of its underlying directed multi-graph (V, E) .

In this definition of the Sabotage Game, *Blocker* removes an edge between two states v, v' by decreasing the value of $E(v, v')$ by 1. As we will see later, this definition of the game based on the above definition of multi-graphs can lead to some technical problems when transforming such a graph into a Sabotage Model. Therefore, we will now present an alternative definition, which we later show (Theorem 1) to be equivalent with respect to the existence of a winning strategy.

Definition 2.3: Let $\Sigma = \{a_1, \dots, a_n\}$ be a finite set of labels. A *directed labelled multi-graph* is a tuple $G^\Sigma = (V, \mathcal{E})$ where V is a set of vertices and $\mathcal{E} = (\mathcal{E}_{a_1}, \dots, \mathcal{E}_{a_n})$, where $\mathcal{E}_{a_i} \subseteq V \times V$ for each $a_i \in \Sigma$.

In this definition, labels from Σ are used to represent multiple edges between two vertices; \mathcal{E} is simply an ordered collection of binary relations on V with labels from Σ . Then, the definition of the game is as follows.

Definition 2.4: A *Labelled Sabotage Game* $SG^\Sigma = \langle V, \mathcal{E}, v, v_g \rangle$ is given by a directed labelled multi-graph (V, \mathcal{E}) and two vertices $v, v_g \in V$. Vertex v represents the position of *Runner* and v_g represents the goal state.

Each match is played as follows: the initial position $\langle \mathcal{E}^0, v_0 \rangle$ is given by $\langle \mathcal{E}, v \rangle$. Round $k + 1$ from position $\langle \mathcal{E}^k, v_k \rangle$ with $\mathcal{E}^k = (\mathcal{E}_{a_1}^k, \dots, \mathcal{E}_{a_n}^k)$, consists of *Runner* moving to some v_{k+1} such that $(v_k, v_{k+1}) \in \mathcal{E}_{a_i}^k$ for some $a_i \in \Sigma$, and then *Blocker* removing an edge $((v, v'), a_j)$, where $(v, v') \in \mathcal{E}_{a_j}^k$ for some $a_j \in \Sigma$. The new position is $\langle \mathcal{E}^{k+1}, v_{k+1} \rangle$, where $\mathcal{E}_{a_i}^{k+1} = \mathcal{E}_{a_i}^k \setminus \{(v, v')\}$ and $\mathcal{E}_{a_i}^{k+1} = \mathcal{E}_{a_i}^k$ for all $i \neq j$. The match ends if a player cannot make a move or if *Runner* reaches the goal state, with him winning only in the last case.

What is said in Remark 1 also holds for Labelled Sabotage Games.

In this definition of the game, it is easy to see that when *Blocker* removes an edge from v to v' , it is irrelevant what is the label of the removed edge; what matters

for the existence of a winning strategy is the number of edges from v to v' that are left.

Observation 1: Let $SG^\Sigma = \langle V, \mathcal{E}, v_0, v_g \rangle$ and $SG'^\Sigma = \langle V, \mathcal{E}', v_0, v_g \rangle$ be two Labelled Sabotage Games that differ only in the labels of their edges, that is,

$$\forall (v, v') \in V \times V : |\{\mathcal{E}_i \mid (v, v') \in \mathcal{E}_i\}| = |\{\mathcal{E}'_i \mid (v, v') \in \mathcal{E}'_i\}|,$$

where $|\cdot|$ stands for cardinality. Then Runner has a winning strategy in SG^Σ iff he has a winning strategy in SG'^Σ .

We will now show that the problems of deciding whether Runner has a winning strategy in each of the Sabotage Games SG and SG^Σ are polynomially equivalent. We start by formalizing the problems.

Definition 2.5: The decision problem SABOTAGE is defined as follows.

- **INPUT:** A Sabotage Game $SG = \langle V, E, v_0, v_g \rangle$.
- **QUESTION:** Does Runner have a winning strategy in SG ?

Definition 2.6: The decision problem Σ -SABOTAGE is defined as follows.

- **INPUT:** A Sabotage Game on a labelled multi-graph $SG^\Sigma = \langle V, \mathcal{E}, v_0, v_g \rangle$.
- **QUESTION:** Does Runner have a winning strategy in SG^Σ ?

Theorem 1: SABOTAGE and Σ -SABOTAGE are polynomially equivalent.

Proof: We show that the problems can be polynomially reduced to each other.

First we show that SABOTAGE can be reduced to Σ -SABOTAGE. Given a Sabotage Game $SG = \langle V, E, v_0, v_g \rangle$, let $m := \max\{E(u, u') \mid (u, u') \in (V \times V)\}$. Define the Labelled Sabotage Game $f(SG) := \langle V, \mathcal{E}, v_0, v_g \rangle$ where $\mathcal{E} := \{\mathcal{E}_1, \dots, \mathcal{E}_m\}$ and each \mathcal{E}_i is given by $\mathcal{E}_i := \{(u, u') \in V \times V \mid E(u, u') \geq i\}$.

We show that Runner has a winning strategy (w.s.) in SG iff he has one in $f(SG)$. The proof is by induction on $n = \sum_{(v, v') \in V \times V} E(v, v')$, which is the number of edges of SG . Note that by definition of f , $n = \sum_{i=1}^{i=m} |\mathcal{E}_i|$, that is, $f(SG)$ has the same number of edges.

The base case is straightforward since in both games Runner has a w.s. iff $v_0 = v_g$. For the inductive case, from left to right, suppose Runner has a w.s. in the game $SG = \langle V, E, v_0, v_g \rangle$ with $n + 1$ edges. Then, there is some $v_1 \in V$ such that $E(v_0, v_1) > 0$ and Runner has a w.s. for all games $SG' = \langle V, E', v_1, v_g \rangle$ that result from Blocker removing any edge (u, u') with $E(u, u') > 0$. Note that all such games SG' have just n edges, so by induction hypothesis Runner has a w.s. in $f(SG')$. But then, by Observation 1, Runner has also a w.s. in all games $f(SG')$ that result from removing an arbitrary edge from $f(SG)$, because for any removed edge (u, u') , the only possible difference between $f(SG')$ and $f(SG)$ is in the labels of the edges between u and u' (in $f(SG')$ the removed label was the largest, in $f(SG)$ the removed label is any). Now,

TABLE II
SABOTAGE LEARNING GAMES

Game	Winning Condition
SLGUE	Learner wins iff he reaches the goal state. Teacher wins otherwise.
SLGHU	Teacher wins iff Learner reaches the goal state. Learner wins otherwise.
SLGHE	Both players win iff Learner reaches the goal state. Both lose otherwise.

by definition of f , choosing v_1 is also a legal move for Runner in $f(SG)$ and, since he can win every $f(SG')$, he has a w.s in $f(SG)$.

From right to left, Runner having a w.s. in $f(SG)$ means that he can choose some v_1 with $(v_0, v_1) \in \mathcal{E}_i$ for some $i \leq m$ such that he has a w.s. in all games $f(SG)'$ resulting from Blocker's move. Choosing v_1 is also a legal move of Runner in SG . Suppose that Blocker replies by choosing (v, v') . Let us call the resulting game SG' . By assumption and Observation 1, Runner also has a w.s. in the game $f(SG')$ which is the result from Blocker choosing $((v, v'), E(v, v'))$. Since $f(SG) = f(SG')$, we can apply the inductive hypothesis.

Let us see now how SG^Σ can be polynomially reduced to SG . Given $SG^\Sigma = \langle V, \mathcal{E}, v, v_g \rangle$ with $\Sigma = \{a_1, \dots, a_m\}$, define $f'(SG^\Sigma) := \langle V, E, v, v_g \rangle$, where $E(v, v') := |\{\mathcal{E}_{a_i} \mid (v, v') \in \mathcal{E}_{a_i}\}|$.

Showing that Runner has a w.s. in SG^Σ iff he has one in $f(SG^\Sigma)$ is straightforward, and can be done by induction on $n := \sum_{a \in \Sigma} |\mathcal{E}_a|$. Both f and f' are polynomial. ■

B. Sabotage Learning Games

Based on the Sabotage Games framework, we define Sabotage Learning Games as follows.

Definition 2.7: A Sabotage Learning Game (SLG) is a Labelled Sabotage Game played by *Learner* (L , taking the role of Runner) and *Teacher* (T , taking the role of Blocker). We distinguish between three different versions, *SLGUE*, *SLGHU* and *SLGHE*, differing in the winning conditions (given in Table II).

The different winning conditions correspond to different levels of Teacher's helpfulness and Learner's willingness to learn. We can have an *unhelpful* teacher and an *eager* learner (*SLGUE*), but there is also the possibility of a *helpful* teacher and an *unwilling* learner (*SLGHU*). The cooperative case corresponds to the version with a *helpful* teacher and an *eager* learner (*SLGHE*).

We now show how Sabotage Modal Logic can be used for reasoning about Learner's and Teacher's strategic power in the learning games previously defined.

C. Sabotage Modal Logic

Sabotage Modal Logic (SML) has been introduced in [11]. Besides the standard modalities, it also contains

“transition-deleting” modalities for reasoning about model change that occurs when a transition is removed. To be more precise, we have formulas of the form $\diamond_a\phi$, expressing that it is possible to delete a pair from the accessibility relation such that ϕ holds in the resulting model at the current state.

Definition 2.8 (Sabotage Modal Language [11]): Let PROP be a countable set of propositional letters and let Σ be a finite set. Formulas of the language of Sabotage Modal Logic are given by

$$\phi ::= p \mid \neg\phi \mid \phi \vee \psi \mid \diamond_a\phi \mid \diamond_a\psi$$

with $p \in \text{PROP}$ and $a \in \Sigma$. We write $\diamond\phi$ for $\bigvee_{a \in \Sigma} \diamond_a\phi$ and $\diamond_a\phi$ for $\bigvee_{a \in \Sigma} \diamond_a\phi$.

Definition 2.9 ([7]): Given a countable set of propositional letters PROP and a finite set $\Sigma = \{a_1, \dots, a_n\}$, a *Sabotage Model* is a tuple $M = \langle W, (R_{a_i})_{a_i \in \Sigma}, \text{Val} \rangle$ where W is a non-empty set of worlds, each $R_{a_i} \subseteq W \times W$ is an accessibility relation and $\text{Val} : \text{PROP} \rightarrow \mathcal{P}(W)$ is a propositional valuation function. The pair (M, w) with $w \in W$ is called a *Pointed Sabotage Model*.

For the semantics of SML, we first define the model that results from removing an edge.

Definition 2.10: Let $M = \langle W, R_{a_1}, \dots, R_{a_n}, \text{Val} \rangle$ be a Sabotage Model. The model $M_{(v,v')}^{a_i}$ that results from removing the edge $(v, v') \in R_{a_i}$ is defined as

$$M_{(v,v')}^{a_i} := \langle W, R_{a_1}, \dots, R_{a_{i-1}}, R_{a_i} \setminus \{(v, v')\}, R_{a_{i+1}}, \dots, R_{a_n}, \text{Val} \rangle.$$

Definition 2.11: Given a Sabotage Model $M = \langle W, (R_a)_{a \in \Sigma}, \text{Val} \rangle$ and a world $w \in W$, atomic propositions, negations, disjunctions and standard modal formulas are interpreted as usual. For the case of “transition-deleting” formulas, we have

$$M, w \models \diamond_a\phi \text{ iff } \exists v, v' \in W : (v, v') \in R_a \ \& \ M_{(v,v')}^a, w \models \phi,$$

and $\exists_a\phi$ is defined to be equivalent to $\neg\diamond_a\neg\phi$.

Theorem 2 ([7]): Combined complexity of model checking for SML is PSPACE-complete.

Note that “combined complexity” means that both the formula and the model are taken as input.

D. Sabotage Learning Games in Sabotage Modal Logic

For any given Sabotage Learning Game SG^Σ we can construct a Pointed Sabotage Model $M(SG^\Sigma)$ in a straightforward way.

Definition 2.12: Let $SG^\Sigma = \langle V, \mathcal{E}, v_0, v_g \rangle$ be a Sabotage Game with $\mathcal{E} = (\mathcal{E}_a)_{a \in \Sigma}$. We define the Pointed Sabotage Model $(M(SG^\Sigma), v_0)$ over the set of atomic propositions $\text{PROP} := \{\text{goal}\}$ with

$$M(SG^\Sigma) := \langle V, \mathcal{E}, \text{Val} \rangle,$$

where $\text{Val}(\text{goal}) := \{v_g\}$.

In the light of this construction, SML becomes useful for reasoning about players’ strategic power in SLGs. For each winning condition in Table II, we can define

a formula of SML that characterizes the existence of a winning strategy, that is, the formula is true in a given Pointed Sabotage Model if and only if the corresponding player has a winning strategy in the game represented by the model.

First we look at the game *SLGUE* (the standard Sabotage Game of [11]), with Learner trying to reach the goal state and Teacher trying to prevent him from doing so. Inductively, we define:

$$\gamma_0^{UE} := \text{goal}, \quad \gamma_{n+1}^{UE} := \text{goal} \vee \diamond\exists\gamma_n^{UE}.$$

Our following result is a variation of Theorem 7 of [7], rephrased for Labelled Sabotage Games. We provide a detailed proof to show how our “labelled” definition avoids a technical issue present in the original proof.

Theorem 3: Learner has a winning strategy in the *SLGUE* game $SG^\Sigma = \langle V, \mathcal{E}^0, v_0, v_g \rangle$ iff $M(SG^\Sigma), v_0 \models \gamma_n^{UE}$, for $n := \sum_{a \in \Sigma} |\mathcal{E}_a^0|$ (the number of edges in (V, \mathcal{E}^0)).

Proof: The proof is by induction on n .

Base case

(\Rightarrow) L having a w.s. in SG^Σ implies that $v_0 = v_g$. Thus, $M(SG^\Sigma), v_0 \models \text{goal}$ and hence, $M(SG^\Sigma), v_0 \models \gamma_0^{UE}$.

(\Leftarrow) $M(SG^\Sigma), v_0 \models \gamma_0^{UE}$ means that $M(SG^\Sigma), v_0 \models \text{goal}$. Thus $v_0 = v_g$. Hence, L wins SG^Σ immediately.

Inductive case

(\Rightarrow) Suppose that SG^Σ has $n+1$ edges, and assume that L has a w.s. There are two possibilities. (1) v_0 is the goal state; then $M(SG^\Sigma), v_0 \models \text{goal}$ and hence $M(SG^\Sigma), v_0 \models \gamma_{n+1}^{UE}$. (2) v_0 is not the goal state. Since L has a w.s., there is some $v_1 \in V$ such that $(v_0, v_1) \in \mathcal{E}_{a_i}^0$ for some $a_i \in \Sigma$ and no matter which pair $((u, u'), a_j) \in (V \times V) \times \Sigma$ with $(u, u') \in \mathcal{E}_{a_j}^0$ T chooses, L has a w.s. in the resulting game $SG'^\Sigma = \langle V, \mathcal{E}^1, v_1, v_g \rangle$, with $\mathcal{E}^1 = (\mathcal{E}_{a_1}^0, \dots, \mathcal{E}_{a_{j-1}}^0, \mathcal{E}_{a_j}^0 \setminus \{(u, u')\}, \mathcal{E}_{a_{j+1}}^0, \dots, \mathcal{E}_{a_{|\Sigma|}}^0)$. Now, SG'^Σ has n edges and thus by inductive hypothesis, $M(SG'^\Sigma), v_1 \models \gamma_n^{UE}$. This implies $M(SG^\Sigma), v_0 \models \diamond\exists\gamma_n^{UE}$ and thus $M(SG^\Sigma), v_0 \models \gamma_{n+1}^{UE}$. (\Leftarrow) $M(SG^\Sigma), v_0 \models \text{goal} \vee \diamond\exists\gamma_n^{UE}$ implies that v_0 is the goal state (so L wins immediately) or else there is v_1 accessible from v_0 such that $M(SG^\Sigma), v_1 \models \exists\gamma_n^{UE}$, that is, $M(SG^\Sigma)_{(v,v')}^{a_i}, v_1 \models \exists\gamma_n^{UE}$ for any $((v, v'), a_i) \in (V \times V) \times \Sigma$. By inductive hypothesis, this gives L a w.s. at v_1 in a game that results from removing any edge from SG^Σ , and hence a w.s. at v_0 in the game SG^Σ . ■

They key observation for the left-to-right direction of this proof is that the model that results from removing an edge from $M(SG^\Sigma)$ is always a model that results from transforming a Labelled Sabotage Game into a model. With the original definition of a Sabotage Game, this is not the case: after removing an edge between v and v' with label k , the resulting model does not need to be the image of a multi-graph because the label of the removed edge does not need to be the biggest of them. Another way to look at it is the following: the multiple edges of the original multi-graph can be seen

as implicitly labelled by numbers, and the existence of an edge labelled with k implies the existence of edges labelled with $1, \dots, k-1$. This property is not preserved when Teacher removes an edge with an arbitrary label from the model $M(SG)$.

Consider now the game $SLGHU$, with Teacher trying to force Learner to reach the goal state. Inductively, define

$$\gamma_0^{HU} := goal, \quad \gamma_{n+1}^{HU} := goal \vee (\diamond \top \wedge \square \diamond \gamma_n^{HU}).$$

Now, we can show that this formula corresponds to the existence of a winning strategy for Teacher. Note that in order to win, Teacher has to make sure that Learner does not get stuck before he has reached the goal state. This is why we need the conjunct $\diamond \top$ in the formula.

Theorem 4: Teacher has a winning strategy in the $SLGUE$ game $SG^\Sigma = \langle V, \mathcal{E}^0, v_0, v_g \rangle$ iff $M(SG^\Sigma), v_0 \models \gamma_n^{HU}$, for $n := \sum_{a \in \Sigma} |\mathcal{E}_a^0|$.

Proof: Similar to the proof of Theorem 3. ■

Finally, consider $SLGHE$, with Teacher and Learner winning iff Learner reaches the goal state. The corresponding formula is defined as follows

$$\gamma_0^{HE} := goal, \quad \gamma_{n+1}^{HE} := goal \vee \diamond \diamond \gamma_n^{HE}.$$

Theorem 5: Teacher and Learner have a joint winning strategy in the $SLGHE$ game $SG^\Sigma = \langle V, \mathcal{E}^0, v_0, v_g \rangle$ iff $M(SG^\Sigma), v_0 \models \gamma_n^{HE}$, for $n := \sum_{a \in \Sigma} |\mathcal{E}_a^0|$.

Proof: Note that L and T have a joint w.s. iff there is a path from v_0 to v_g . From left to right this is obvious. From right to left, if there is such path, then there is also one without cycles; then, there is a joint w.s. that follows the path and at each step removes the edge that has just been used. The Theorem follows by observing that γ_n^{HE} expresses the existence of such path. ■

The previous results are summarized in Table III.

TABLE III
WINNING CONDITIONS FOR SLG IN SML

Game	Winning Condition in SML	Winner
$SLGUE$	$\gamma_0^{UE} := goal, \gamma_{n+1}^{UE} := goal \vee \diamond \exists \gamma_n^{UE}$	Learner
$SLGHU$	$\gamma_0^{HU} := goal, \gamma_{n+1}^{HU} := goal \vee (\diamond \top \wedge (\square \diamond \gamma_n^{HU}))$	Teacher
$SLGHE$	$\gamma_0^{HE} := goal, \gamma_{n+1}^{HE} := goal \vee \diamond \diamond \gamma_n^{HE}$	Both

E. Complexity of Sabotage Learning Games

Intuitively, some versions of the Sabotage Learning Game are simpler than others. With a helpful teacher and an eager learner, the learning process should be easier than with an unhelpful teacher or a unwilling learner. This is indeed reflected in the computational complexity of deciding in a given game whether the winning condition is satisfied.

We have shown that our three winning conditions (Table III) can be expressed in SML, and Theorem 2

(proved in [7]) tells us that model checking of SML is PSPACE-complete. This gives us PSPACE upper bounds for the complexity of the problems of deciding whether each winning condition is satisfied in a given game. For two of the winning conditions ($SLGUE$ and $SLGHE$), we can also give tight lower bounds.

For $SLGUE$ – the standard Sabotage Game – PSPACE-hardness is shown by reduction from QBF [7].

Theorem 6 ([7]): $SLGUE$ is PSPACE-complete.

As mentioned above, for $SLGHU$ we obtain a PSPACE upper bound.

Theorem 7: $SLGHU$ is in PSPACE.

Proof: Follows from Theorem 2 and Theorem 4. ■

It remains to be shown whether $SLGHU$ is also PSPACE-hard. Whereas at first sight, $SLGHU$ and $SLGUE$ might seem to be duals of each other, the relationship between them is more complex due to the different nature of the players' moves (Learner moves locally by choosing an accessible state, whereas Teacher moves globally, manipulating the structure in which Learner moves). Thus, a reduction from $SLGUE$ to $SLGHU$ is not straightforward. Let us now look at $SLGHE$. This game is of a different nature than the two previous ones. It is cooperative, and a winning strategy is a joint strategy for both players. Such a strategy does not need to take into account all possible moves of the opponent. This suggests that this version should be less complex than $SLGUE$ and $SLGHU$.

The following result shows that at least for the comparison of $SLGUE$ and $SLGHE$, this is indeed the case: for an eager learner, learning with a helpful teacher is easier than learning with an unhelpful one. This follows from the fact that the winning condition of $SLGHE$ is satisfied iff the goal vertex is reachable from the initial vertex (note that Learner moves first). Thus, determining whether Teacher and Learner can win $SLGHE$ is equivalent to solving the REACHABILITY (st-CONNECTIVITY) problem, which is known to be nondeterministic logarithmic space (NL)-complete [9].

Theorem 8: $SLGHE$ is NL-complete.

Proof: Polynomial equivalence of $SLGHE$ and REACHABILITY follows from the argument given in the proof of Theorem 5. ■

Table IV summarizes the complexity results for the different versions of SLG .

TABLE IV
COMPLEXITY RESULTS FOR SABOTAGE LEARNING GAMES

Game	Winning Condition	Complexity
$SLGUE$	Learner wins iff he reaches the goal state, Teacher wins otherwise	PSPACE-complete.
$SLGHU$	Teacher wins iff Learner reaches the goal state, Learner wins otherwise.	PSPACE
$SLGHE$	Both players win iff Learner reaches the goal state. Both loose otherwise.	NL-complete

In the case of an eager Learner, the complexity results agree with our intuitions when comparing the cooperative version of the Sabotage Game (*SLGHE*) with the non-cooperative one (*SLGUE*). The easiest way to learn for an eager Learner is when the Teacher is helpful.

III. RELAXING STRICT ALTERNATION

As mentioned above, Learner's moves in the graph are interpreted as changes of information states. Then, when Teacher removes an edge, she actually informs Learner which changes of information state should not be performed. In this perspective, Learner's moves can be seen as internal ones while Teacher's moves can be interpreted externally. Due to this asymmetry, each Learner's move does not in principle need to be followed by a teacher's move.

Definition 3.1: A *Sabotage Learning Game without strict alternation* (for Teacher) is a tuple $SLG^* = \langle V, \mathcal{E}, v_0, v_g \rangle$. Moves of Learner are as in the Sabotage Learning Game and, once he has chosen a vertex v_1 , Teacher has a choice between removing an edge, in which case the next game is given as in *SLG*, and doing nothing, in which case the next game is $\langle V, \mathcal{E}, v_1, v_g \rangle$. We again distinguish between three versions, SLG^*UE , SLG^*HU and SLG^*HE , with winning conditions given as before.

Though we provide Teacher with an additional possible move, this does not change her winning abilities. In the rest of this section we show that, for the three variations of a Sabotage Learning Game, a player has a w.s. in SLG^* iff she has a w.s. in *SLG*.

Consider the case of an unhelpful teacher and an eager learner SLG^*UE . Before we go into the details, note that if Learner can win the game, he can do so in a finite number of rounds.

Theorem 9: Consider the *SLG* $\langle V, \mathcal{E}, v_0, v_g \rangle$ with (V, \mathcal{E}) a directed labelled multi-graph and v, v_g vertices in it. If Learner has a winning strategy in the corresponding *SLGUE*, then he has a winning strategy in the corresponding SLG^*UE .

Proof: This can be shown by induction on the number of rounds. The idea is that in each round L "pretends" that T has removed some edge and then makes the move given by his strategy for *SLGUE*. ■

If L can win a SLG^*UE , then it is easy to see that he can also win the corresponding *SLGUE* by using his w.s. from SLG^*UE .

Corollary 1: Consider the tuple $\langle V, \mathcal{E}, v_0, v_g \rangle$ with (V, \mathcal{E}) a directed labelled multi-graph and v, v_g vertices in it. Learner has a winning strategy in the corresponding SLG^*UE iff he has a winning strategy in the corresponding *SLGUE*.

The case of a helpful teacher and an unwilling learner is more interesting. One might expect that the additional possibility of an empty move gives more power to Teacher since it allows her to skip a move in cases

when removing an edge would have made the goal unreachable from the current vertex. However, we can show that this is not the case. First, we state the following lemmas.

Lemma 1: For any $SLG^*HU \langle V, \mathcal{E}, v_0, v_g \rangle$, if there is a path from v_0 to v_g and there is no path from v_0 to a state from where v_g is not reachable, then T has a winning strategy.

Proof: By assumption, all states reachable from v_0 are on paths to v_g . Therefore, even if T will refrain from removing any edge, L will always be on some path to the goal. There are two possibilities: either the path to the goal does not include a loop or it does. If it does not then T can simply wait until L will arrive to the goal. If it does, in order to win T can remove the edges that lead into the loops in such a way that v_g is still reachable from any vertex. L will eventually have to move to v_g . ■

Lemma 2: Consider the SLG^*HU game $\langle V, \mathcal{E}, v_0, v_g \rangle$. If T has a winning strategy and there is some edge $(v, v') \in \mathcal{E}_a$ for some $a \in \Sigma$ such that no path from v_0 to v_g goes via the edge (v, v') , then T also has a winning strategy in $\langle V, \mathcal{E}', v_0, v_g \rangle$, where \mathcal{E}' is the result of removing (v, v') from \mathcal{E}_a .

Proof: If v is not reachable from v_0 , it is easy to see that the claim holds. Let us consider the case that v is reachable from v_0 . Since there is no path to v_g visiting v , T 's winning strategy should keep L away from it (otherwise L would win). Hence, T can also win if the edge (v, v') is not there. ■

Theorem 10: If Teacher has a winning strategy in the $SLG^*HU \langle V, \mathcal{E}, v_0, v_g \rangle$, then she also has a winning strategy in which she removes an edge in each round.

Proof: The proof proceeds by induction on the number of edges $n = \sum_{a \in \Sigma} |\mathcal{E}_a|$.

The base case is straightforward. For the inductive case, assume that T has a winning strategy in $SLG^*HU \langle V, \mathcal{E}, v_0, v_g \rangle$ with $\sum_{a \in \Sigma} |\mathcal{E}_a| = n + 1$.

Then if $v_0 = v_g$, we are done. Thus, assume that $v_0 \neq v_g$. Then, since T can win, there is some $v_1 \in V$ such that $(v_0, v_1) \in \mathcal{E}_a$ for some $a \in \Sigma$ and for all such v_1 it holds that:

- 1) There is a path from v_1 to v_g , and
- 2) a) T can win $\langle V, \mathcal{E}, v_1, v_g \rangle$, or
b) there is some $((v, v'), a) \in (V \times V) \times \Sigma$ such that $(v, v') \in \mathcal{E}_a$ and T can win $\langle V, \mathcal{E}', v_1, v_g \rangle$ where \mathcal{E}' is the result of removing (v, v') from \mathcal{E}_a .

If 2b holds, since $\sum_{a \in \Sigma} |\mathcal{E}'_a| = n$, we are done — we can use the inductive hypothesis and conclude that T has a w.s. in which she removes an edge in each round (in particular, she chooses $((v, v'), a)$ in the first round). This $((v, v'), a)$ can be chosen in one of the following ways.

If there is some $(v, v') \in V \times V$ such that $(v, v') \in \mathcal{E}_a$ for some $a \in \Sigma$ and this edge is not part of any path from v_1 to v_g then by Lemma 2, T can remove this edge and 2b holds and we are done.

If every edge in (V, \mathcal{E}) belongs to some path from v_1 to v_g , from 1, there are two cases: either there is only one, or there are more than one paths from v_1 to v_g .

In the first case (only one path) (v_0, v_1) can be chosen since it cannot be part of the *unique* path from v_1 to v_g .

Assume now that there is more than one path from v_1 to v_g . Let $p = (v_1, v_2, \dots, v_g)$ be the/a shortest path from v_1 to v_g . This path cannot contain any loops. Then, from this path take v_i such that i is the smallest index for which it holds that from v_i there is a path $(v_i, v'_{i+1}, \dots, v_g)$ to v_g that is at least as long as the path following p from v_i (i.e. $(v_i, v_{i+1}, \dots, v_g)$). Intuitively, when following path p from v_1 to v_g , v_i is the first point at which one can deviate from p in order to take another path to v_g (recall that we consider the case where every vertex in the graph is part of some path from v_1 to v_g). Now it is possible for T to choose $((v_i, v'_{i+1}), a)$ such that $(v_i, v'_{i+1}) \in \mathcal{E}_a$. Let \mathcal{E}' be the resulting set of edges after removing (v_i, v'_{i+1}) from \mathcal{E}_a . Then we are in the position $\langle V, \mathcal{E}', v_1, v_g \rangle$. Note that because of the way we chose the edge that has been removed, in the new graph it still holds that from v_0 there is no path to a vertex from which v_g is not reachable (this holds because from v_i the goal v_g is still reachable). Then by Lemma 1, T can win $\langle V, \mathcal{E}', v_1, v_g \rangle$, which then implies 2b.

Hence, we conclude that 2b has to be the case and thus using the inductive hypothesis, we conclude that T can win the game $\langle V, \mathcal{E}, v_0, v_g \rangle$ also by removing an edge in every round. ■

Corollary 2: Consider the tuple $\langle V, \mathcal{E}, v_0, v_g \rangle$ with (V, \mathcal{E}) a directed labelled multi-graph and v_0, v_g vertices in it. Teacher has a winning strategy in the corresponding SLG^*HU , iff she has a winning strategy in the corresponding $SLGHU$.

Finally, let us move to the case of a helpful teacher and an eager learner.

Theorem 11: Consider the tuple $\langle V, \mathcal{E}, v_0, v_g \rangle$ with (V, \mathcal{E}) a directed labelled multi-graph and v_0, v_g vertices in it. If Learner and Teacher have a winning strategy in the corresponding SLG^*HE , then they have a winning strategy in the corresponding $SLGHE$.

Proof: The proof of Theorem 5 provides the needed strategy. ■

Corollary 3: Consider the tuple $\langle V, \mathcal{E}, v_0, v_g \rangle$ with (V, \mathcal{E}) a directed labelled multi-graph and v, v_g vertices in it. Learner and Teacher have joint winning strategy in the corresponding SLG^*HE iff they have a joint winning strategy in the corresponding $SLGHE$.

In this section we have shown that in Sabotage Learning Games, allowing Teacher to skip moves, does not change the winning abilities of the players. Using these results, both the complexity and definability results from the previous section also apply to the versions of the game in which Teacher can refrain from making a move.

IV. REFINED VIEW ON TEACHING: LEARNING ALGORITHMS

The perspective on learning that we have adopted is very general. To give a more refined view, let us go back to the *queries and counterexamples* paradigm (see [1]). In that approach, Learner is an algorithm that embodies a winning strategy in the game of learning (the learning procedure succeeds on *all possible true data*). Teacher can significantly influence the learning process by giving counterexamples, and the time needed for learning depends on her choices. Therefore, the game of teaching in such a setting can be formalized in extensive form as presented in Figure 1.

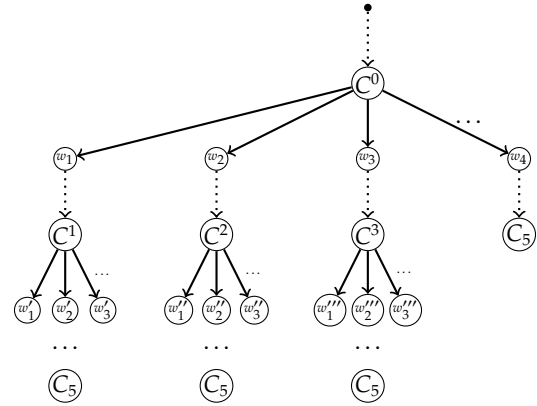


Fig. 1. The tree of the teaching game: dotted lines are Learner's moves, which are determined by his algorithm; solid lines are Teacher's moves; w_i are counterexamples given by Teacher; C_i are conjectures made by Learner; C_5 is the correct hypothesis.

There are many game-theoretical issues that arise when viewing the run of the learning algorithm as a game. We can for example consider the epistemic status of the players, introduce imperfect information and analyze payoff characteristics. Concerning the payoff characteristics and different classes of teachers such as (un)helpful teachers, we can define corresponding preference relations or payoffs: the *helpful* teacher may strictly prefer all shortest paths in the game tree, i.e. the paths in which the learner learns the fastest. The *unhelpful* teacher might strictly prefer all the longest paths in the game tree, i.e. the paths in which the learner learns slowly.

We can also provide a choice for Learner in this game. Firstly, we can allow that at each step the learner can choose from one or more procedures which are part of one algorithm. Secondly, in the beginning Learner can decide with which of the available algorithms he is going to proceed. Moreover, we can consider also another possibility that involves extending the traditional inductive inference paradigm. Usually, learnability of a class is interpreted as the existence of a learner that learns every element from the class independently of the behavior of Teacher — if we introduce the possibility of non-learnability to the game, we can view learning algorithms as winning strategies for an eager learner in the

learning game. With the possibility of non-learnability, there are also paths in the game tree in which the learner never makes a correct conjecture. In this framework, a *helpful* teacher would also prefer all (shortest) paths ending in a position in which the learner makes a correct conjecture over all the other paths. An *unhelpful* teacher then prefers all the paths in which the learner does not learn over those in which he does learn.

V. CONCLUSIONS AND FURTHER WORK

We have provided a game theoretical approach to learning that allows us to analyze different levels of cooperativeness between Learner and Teacher. We have defined *Sabotage Learning Games* with three variations, representing different didactic scenarios. Then, we have shown how *Sabotage Modal Logic* can be used to reason about these games and, in particular, we have identified certain formulas of the language with the existence of a winning strategy. We gave complexity results for the decision problems associated with each version of the game. These problems correspond to model checking for the associated formulas and models. Our complexity results support the intuitive claim that the cooperation of agents facilitates learning. We investigated an extension of the Sabotage Learning Games that relaxes the condition of the strict alternation of moves. Our results presented in Section III show that if we allow Teacher to skip a move, the winning abilities of the players do not change with respect to the original versions of the games. In the case of the helpful teacher and unwilling learner, this is quite surprising since it says that if Teacher can force Learner to learn in the game with nonstrict alternation, then even if she is forced to remove edges in each round she can do so without removing edges that are necessary for Learner to eventually reach the goal state.

From the perspective of Formal Learning Theory, several relevant extensions can be done. We have described the learning process as *changes in information states*, without going further into their epistemic and/or doxastic interpretation. A deeper analysis can give us insights about how the learning process is related to different notions of dynamics of information, such as belief revision or dynamic epistemic logic.

Moreover, it can be argued that in some natural learning scenarios, e.g. language learning, the goal of the learning process, e.g. a correct grammar, is concealed from Learner. In our approach we assume that Learner has at least some markers of the goal state, otherwise we could not introduce the two possible characteristics of Learner. The assumption of the absence of any ability to recognize the goal by Learner, leads to a model in which the Learner moves randomly. Some approaches that take into account randomness in Sabotage Games have already been introduced [8] and hopefully can be extended to deal with the aforementioned issue. What

we can now hypothesize is that the complexity of the scenario with a random Learner and a helpful Teacher is bounded by the worst case scenario, in which Learner avoids the goal as long as possible, i.e. the game *SLGHU*.

In the introduction we described the concepts of *finite identification* and *identification in the limit*. Our work on *SLGs* is closer to the first one, as we understand learning as the ability to reach an appropriate information state, without taking into account what will happen after such a state has been reached. In particular, we are not concerned with the stability of the resulting belief. *Identification in the limit* extends *finite identification* by looking beyond reachability in order to describe “ongoing behaviour”. Fixed-point logics, like the propositional μ -calculus [10], [5], can provide us with tools to express this notion of learnability. In this case, epistemic and doxastic interpretations of learning would involve notions of stable belief and a kind of operational, non-introspective knowledge as a result of the process.

REFERENCES

- [1] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.
- [2] H. van Ditmarsch, W. van der Hoek, and B. Kooi. *Dynamic Epistemic Logic*. Springer Netherlands, 2007.
- [3] E.M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
- [4] Sanjay Jain, Daniel Osherson, James S. Royer, and Arun Sharma. *Systems that Learn*. MIT Press, Chicago, 1999.
- [5] Dexter Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
- [6] Christof Löding and Philipp Rohde. Solving the Sabotage Game is PSPACE-hard. In *Proceedings of the 28th International Symposium on Mathematical Foundations of Computer Science, MFCS '03*, volume 2474 of LNCS, pages 531–540. Springer, 2003.
- [7] Christof Löding and Philipp Rohde. Solving the Sabotage Game is PSPACE-hard. Technical report, Aachener Informatik Berichte, Rwth Aachen, 2003.
- [8] Dominik Klein, Frank G. Radmacher, and Wolfgang Thomas. The Complexity of Reachability in Randomized Sabotage Games. Proceedings of FSEN 2009, LNCS, 2009.
- [9] Christos M. Papadimitriou. *Computational complexity*. Addison-Wesley, MA, 1994.
- [10] Dana Scott and J. W. de Bakker. A theory of programs. Unpublished manuscript, IBM, Vienna, 1969.
- [11] Johan van Benthem. An essay on sabotage and obstruction. In *Mechanizing Mathematical Reasoning, Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday*, pages 268–276, 2005.

ACKNOWLEDGMENT

The authors would like to thank Jakub Szymanik and Johan van Benthem for their help and their useful comments on previous versions.

Fernando R. Velázquez-Quesada is supported by **Consejo Nacional de Ciencia y Tecnología (CONACyT)**, México (scholarship # 167693).

On the Incentive Compatible Core of a Procurement Network Formation Game with Incomplete Information

Chandrashekar T S, Narahari Y

Abstract—In this paper we present a model of the multiple unit, single item procurement network formation problem in environments with incomplete information (MPNFI). For this we first develop the structure of the procurement network formation problem within Myerson’s framework for cooperative games with incomplete information [1]. Using this framework we then investigate the non-emptiness of the incentive compatible core, an extension of the notion of the core for complete information settings based on Myerson’s framework, and show that it is indeed non-empty for the class of MPNFI games.

Index Terms—Cooperative Games, Incomplete Information, Procurement Networks

I. INTRODUCTION

THE problem that we consider in this paper is the following: We have a buyer who is interested in buying multiple units of a single item. He associates a certain value for each unit of the item. The item goes through many stages of value addition through a linear supply chain. For each stage of value addition, there is at least one supplier. Each supplier has his own cost of value addition in each of the stages that he is present and also has a limited amount of capacity. The buyer’s valuation and the suppliers’ costs are assumed to be private information. The buyer and the suppliers can enter into a negotiation to finalize an outcome that indicates the number of units that would be produced, the suppliers who would be engaged in the value addition process, and a division of the surplus that accrues from the transaction. In a situation where information is not privately held a division of surplus could be done such that the allocations are in the core of the induced cooperative game. In the incomplete information situation however, this notion of the core needs to be extended appropriately. In doing this there are three basic issues to be considered as to how the agents’ information may be used.

- 1) First, in evaluating whether a coalition can make all its members better off, we must be sure about when the agents’ welfare should be evaluated. There are three stages - ex-ante, interim and ex-post, when this evaluation may be carried out. The appropriate stage to evaluate the welfare of agents in the MPNFI problem is the interim stage when each agent has learnt his own private type information but not that of the other agents.

This work was carried out while the first author was a student at the Indian Institute of Science. E-mail: chandrashekar.ts@gmail.com. Narahari Y is with the Department of Computer Science and Automation at the Indian Institute of Science, Bangalore, INDIA. E-mail: hari@csa.iisc.ernet.in

- 2) Secondly, in defining what a coalition can do, we assume that the agents will use all the information that is available to all its members in deciding upon a blocking plan which leads to the definition of a *fine core* [2].
- 3) Finally, when we assume that agents share information, we must specify whether such communication between agents is verifiable or not. It is natural to see that the private information (valuation and costs) of the agents (buyer and suppliers) is not inherently verifiable and hence incentive constraints need to be incorporated into any analysis of the MPNFI problem.

To summarize, the solution concept that we focus upon for the MPNFI problem is the *interim, incentive compatible fine core*.

A. Interim Incentive Compatible Fine Core

The extension of the core to a cooperative game with incomplete information that we consider for the MPNFI problem is a generalized version of the NTU game involving the method of fictitious transfers of weighted utilities. With incomplete information the role of weighted utility transfers is taken on by *virtual utility*. Virtual utility is defined by a formula that takes informational incentive constraints into account (see papers by Myerson [3], [4], [5]).

B. Contributions of the paper

Our specific contributions in this paper are twofold and are as follows:

- We believe that this is the first attempt to model the procurement network formation problem as a cooperative game within the context of an incomplete information environment.
- We develop the structure of the game based on Myerson’s approach [1] and focus on the incentive compatible fine core as a relevant solution concept to be used for this game. Our main result is to show that the incentive compatible fine core of MPNFI game is non-empty.

II. THE MPNFI PROBLEM

For the scenario introduced in Section I, we would expect the buyer and the suppliers to enter into negotiations to find the *best* way of forming the procurement network. The notion of *best* here includes (a) an efficiency criterion - selecting a set of suppliers and a quantity to be procured that maximizes

the surplus and (b) an equity criterion - sharing the surplus such that the procurement network that is formed remains stable. Clearly the agents would like to negotiate a contract that achieves these objectives. Such a contract would be a state dependent contract. Our problem in this paper is to know whether such a contract exists.

A. Formulation of the MPNFI problem

We let the feasible network for forming the multiple unit single item procurement network be a directed graph $G = (V, E)$ with V as the set of vertices, two special nodes v_o (origin vertex) and v_t (terminal vertex), and $E \subseteq V \times V$ as the set of edges. With each of the edges $e \in E$ we associate the numbers $c(e)$, $l(e)$, and $u(e)$. $c(e)$ represents the cost, $l(e)$ the lower bound on the capacity of the edge, and $u(e)$ the upper bound on the capacity of the edge, respectively. Now, assume that each of the edges is owned by an agent $i \in N$ where N is a finite set of agents $N = \{1, \dots, n, n+1\}$. The agents $\{1, 2, \dots, n\}$ own edges in the network and agent $(n+1)$ is the buyer. That is, we let $\psi : E \rightarrow N$ such that $\psi(e) = i$ implies that agent i owns (possesses) edge e . We let $\mathcal{I}(j)$ and $\mathcal{O}(j)$ represent the set of all incoming and outgoing edges at vertex $j \in V$.

We let E_S represent the set of edges owned by agents in S . We also designate F_S as the flow in the network between the two special nodes v_o and v_t using only the edges E_S that are owned by agents in S . For any flow F_S , we denote the set of owners of the edges that facilitate the flow F_S as $\psi(F_S)$. We assume that if multiple units of the item are available to the buyer by using the flow F_S , then it costs $c(F_S)$ and the buyer is willing to compensate the edge owners with a value bF_S where b is the value that the buyer attaches to a single unit of the item. The surplus from such a transaction is $bF_S - c(F_S)$. We now follow the structure presented in [5] to model the MPNFI scenario as a cooperative game with incomplete information.

1) *Agents and their Resources:* For simplicity of exposition, we assume here that each agent owns one edge in the network. The analysis however can be extended to scenarios where each agent owns multiple edges. We treat the edges and the money that is owned by agents as resources that are to be traded. Each agent $i \in N$ has an initial resource vector $r_i^0 \in \mathbb{R}_+^{n+1}$ where $r_{i,j}^0 \in \{0, 1\}, \forall j \in \{1, 2, \dots, n\}$ and $r_{i,(n+1)}^0 \in \mathbb{R}_+$. This implies that when agent i owns the edge $j \in E$ then $r_{ij}^0 = 1$ and is otherwise 0. Having assumed that there is a one-to-one correspondence between the edges and the agents, we have $r_{i,i}^0 = 1$ and $r_{i,j}^0 = 0, \forall j \neq i$. In addition the $(n+1)^{th}$ entry in the endowment vector r_i^0 indicates the amount of money that agent i has.

2) *Type Information of Agents:* We now specify the private information (costs and valuations) of the agents through the notion of types as introduced in [6]. For any agent $i \in N$ we let T_i denote the set of possible types. For the MPNFI problem we assume that the type refers to one of two pieces of information. The type $t_i \in T_i$ for all edge owning agents $i \in N \setminus \{(n+1)\}$ is a description of the cost that is incurred when an edge is used for an unit amount of flow and for buying agent $(n+1)$ it describes the valuation for a single unit of the item.

With $N = \{1, 2, \dots, n, n+1\}$ as the finite set of agents, we let $T = T_N = \times_{i \in N} T_i$ be the set of all type profiles of all the agents in the game. An information state of the MPNFI scenario is given by $t \in T$ and also written as $t = (t_{-i}, t_i)$ where the notation $-i$ denotes $N \setminus \{i\}$. Similarly, (t_{-i}, \hat{t}_i) denotes the vector t where $t_{-i} = (t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_{n+1})$ and the i^{th} component t_i is changed to $\hat{t}_i \in T_i$. Similarly, $T_{-i} = \times_{j \neq i} T_j$, and for any coalition S , a non-empty subset of N , we let $T_S = \times_{i \in S} T_i$ so that any $t_S \in T_S$ denotes a combination of types $(t_i)_{i \in S}$. We also let \mathcal{C} denote the set of all possible coalitions or non empty subsets of N , that is, $\mathcal{C} = \{S | S \subseteq N, S \neq \emptyset\}$.

Now, for each possible type $t_i \in T_i$, we let $q_i(t_i)$ denote the probability that agent i is of type t_i and we assume that there is probability that the agent is of any one of the types in T_i is positive. That is $q_i(t_i) > 0, \forall t_i \in T_i$. We assume that the agents' types are independent random variables and hence we can write the following:

- $q(t_S) = \times_{i \in S} q_i(t_i), \forall S \subseteq N, \forall t_S \in T_S.$
- $q(t_{-i}) = \times_{j \in N-i} q_j(t_j), \forall i \in N, \forall t \in T.$
- $q(t) = \times_{j \in N} q_j(t_j), \forall t \in T.$

3) *Outcome Sets:* Now, for any subset $S \in \mathcal{C}$, which includes the agent $(n+1)$, we define a set of market transactions. Note that without the buying agent being a part of the coalition, no transaction is possible. The market transaction follows from a surplus maximizing flow computation using the network flow model described earlier in the section. This computation is carried out when the types $t_i \in T_i$ are declared by the agents $i \in S$. We call this set of market transactions as the set of possible outcomes $X_S(t_S)$, such that $X_S(t_S) = \{(r_i)_{i \in S} | r_i \in \mathbb{R}_+^{n+1} \text{ and } \sum_{i \in S} r_{ij} \leq \sum_{i \in S} r_{ij}^0, \forall j \in \{1, 2, \dots, n, n+1\}\}$, where r_i is the outcome vector of agent i after the transaction is carried out. The outcome set specifies that the reallocation of resources and money is such that there is no infusion of additional resources into the system. We also define the set X_S and the set X as the sets that include the outcomes for all possible type declarations $t_S \in T_S$ and all possible coalitions $S \in \mathcal{C}$ respectively. So, $X_S = \bigcup_{t_S \in T_S} X_S(t_S)$ and $X = \bigcup_{S \in \mathcal{C}} X_S$.

The reallocation of resources, i.e., the edges and the money, is carried out as follows: Given the set of edges owned by the agents in S , the capacities on these edges, the edge costs declared by them and the valuation declared by the buyer, a surplus maximising network flow computation identifies the set of edges and edge capacities whose ownership is to be transferred to the buying agent. Following this, each edge agent whose edge is transferred to the buying agent is compensated according to the declared cost. The entire surplus, defined as the difference between the buyer's valuation for the entire flow and the cost incurred by the edge agents in maintaining this flow, that results from the transaction is then given to either the buying agent or to one of the agents who plays an active role in providing the surplus maximising flow.

4) *Utility Functions:* Now, for any outcome $x \in X$ and any $t \in T$, we let the utility for an agent $i \in N$ be $u_i(x, t)$. For any agent i and outcome x , the final outcome vector r_i reflects the edges that it currently owns and the money that it

has after the transfers have been carried out. That is $r_{i,i}$ can be either 0 or 1 and $r_{i,(n+1)} \in \mathfrak{R}$.

So, the utility that an agent $i = N \setminus \{(n+1)\}$ receives from outcome x when his type is t_i is given by:

$$u_i(x, t) = r_{i,(n+1)} + (r_{i,i} - r_{i,i}^0)t_i. \quad (1)$$

The payoff that the buying agent $(n+1)$ gets from an outcome x , when x_{v_t} is the number of units of the item that he gets, and his type is $t_{(n+1)}$, is given by:

$$u_{(n+1)}(x, t) = x_{v_t}t_{(n+1)} + r_{(n+1),(n+1)} - r_{(n+1),(n+1)}^0. \quad (2)$$

5) *Representation of the MPNFI Game:* The MPNFI scenario can now be described by the structure $\Gamma = (X, x^*, (T_i)_{i \in N}, (u_i)_{i \in N}, (q_i)_{i \in N})$.

Here, X refers to the set of all outcomes for all coalitions $S \in \mathcal{C}$ that could be formed; x^* is a default outcome that results when the agents are unable to come to an agreement over the solution. In the context of the MPNFI problem, the default outcome is a null transaction whose utility for all types of all agents is 0. T_i , u_i , and q_i are as defined earlier. This structure Γ of the game is assumed to be known to all agents. In addition we assume that each agent knows his own type before the start of negotiations. We now need to develop a solution to this cooperative game.

III. STATE CONTINGENT CONTRACTS

We assume here that the state contingent contract will be implemented by an external trustworthy mediator who can make side-payments to the agents. A state contingent contract is now defined as follows.

Definition 3.1: A state contingent contract is represented by a pair of functions $(\mu : T \rightarrow \Delta(X), \chi : T \rightarrow \mathfrak{R}^{|N|})$ where $\mu(x|t)$ represents the probability of choosing the outcome $x \in X$ when the agents' types are t and $\chi(t)$ denotes the net monetary side-payments that the mediator makes to agent i when the agents' types are t .

If a mediator proposes to implement such a state contingent contract (μ, χ) , then the agents must evaluate how they would fare if they agreed to its implementation. This evaluation is carried out by the agents at the interim stage and hence the correct measure of evaluation is conditionally expected utilities, conditioned on their private information.

A. Conditionally Expected Utilities

The conditionally expected utility of agent i if he were to agree to participate in the state contingent contract (μ, χ) proposed by a trustworthy mediator is given by:

$$U_i(\mu, \chi|t_i) = \sum_{t_{-i} \in T_{-i}} q(t_{-i})[\chi_i(t) + \sum_{x \in X} \mu(x|t)u_i(x, t)] \quad (3)$$

Now, if agent i is of type t_i but pretends to be of type \hat{t}_i when he reports his type to the mediator who is implementing the state contingent contract (μ, χ) , then his expected utility is given by:

$$U_i(\mu, \chi, \hat{t}_i|t_i) = \sum_{t_{-i} \in T_{-i}} q(t_{-i})[\chi_i(t_{-i}, \hat{t}_i) + \sum_{x \in X} \mu(x|t_{-i}, \hat{t}_i)u_i(x, t)] \quad (4)$$

B. Incentive Feasible Contracts

If a trustworthy mediator were to implement the state contingent contract (μ, χ) by asking all the agents to reveal their types confidentially to him, then each of the agents would find it in their best interest to report their types honestly if and only if the contract (μ, χ) was incentive compatible. That is, conditionally expected utilities of the agents satisfy the following inequality:

$$U_i(\mu, \chi|t_i) \geq U_i(\mu, \chi, \hat{t}_i|t_i), \quad \forall t_i \in T_i, \forall \hat{t}_i \in T_i, \forall i \in N \quad (5)$$

Since we have assumed that the mediator makes side-payments to the agents, the expected utility for the mediator from implementing the incentive compatible contract (μ, χ) is $-\sum_{t \in T} q(t) \sum_{i \in N} \chi_i(t)$. So, if we want a state contingent contract that is implementable, we should then look for one that is (a) incentive compatible and (b) gives the mediator non-negative utility so that he does not lose from implementing the mechanism.

The utility that the mediator gets from implementing the state contingent contract (μ, χ) is equal to $-\sum_{t \in T} q(t) \sum_{i \in N} \chi_i(t)$. So we want the following inequality to be satisfied.

$$\sum_{t \in T} q(t) \sum_{i \in N} \chi_i(t) \leq 0 \quad (6)$$

Formally, we call such state contingent contracts as *incentive feasible contracts* and we define these as follows.

Definition 3.2: We say that a state contingent contract is incentive feasible if and only if it is incentive compatible and yields a non-negative expected payoff to the mediator. That is, it satisfies the inequalities 5 and 6.

In general, we know that there are a number of such incentive feasible state contingent contracts. The mediator's problem is to pick one such state contingent contract to implement. It would therefore be useful if he could be guided by the same criteria of efficiency and equity, but with appropriate extensions, in evaluating the incentive feasible state contingent contract to implement.

IV. THE EFFICIENCY PRINCIPLE

We have seen that in cooperative games with incomplete information, the appropriate object over which negotiations are carried out is the interim incentive compatible state contingent contract. And since conditionally expected utility is the appropriate measure of welfare evaluation of the agents, the mediator would be well placed in selecting a state contingent contract that maximizes the sum of conditionally expected utilities of the agents in the MPNFI game. We call such a contract an *incentive-efficient contract*. Formally, it is defined as follows:

Definition 4.1: A state contingent contract (μ, χ) is weakly incentive-efficient if and only if it is *incentive feasible* and no other feasible state contingent contract yields higher expected utilities for all types of all agents.

So, we are interested in choosing a state contingent contract (μ, χ) that maximizes the conditionally expected utilities of all agents from among all contracts that obey inequalities (5) and (6). It is easy to see that the incentive constraints specified by (5) are convex. So, from convexity of the incentive constraints and linear programming theory, we can say that a feasible state contingent contract $(\hat{\mu}, \hat{\chi})$ is incentive efficient if and only if there exists some vector $\lambda = (\lambda_i(t_i))_{t_i \in T_i, i \in N}$ such that $\lambda_i(t_i) \geq 0, \forall t_i \in T_i, \forall i \in N$ with at least one strict inequality and $(\hat{\mu}, \hat{\chi})$ maximizes $\sum_{i \in N} \sum_{t_i \in T_i} \lambda_i(t_i) U_i(\mu, \chi | t_i)$ over all feasible state contingent contracts (μ, χ) (See [5]). This is a linear programming problem in (μ, χ) .

$$\begin{aligned} & \text{Maximize} && \sum_{i \in N} \sum_{t_i \in T_i} \lambda_i(t_i) U_i(\mu, \chi | t_i) \\ & \text{s.t.} && \\ & U_i(\mu, \chi | t_i) \geq && U_i(\mu, \chi, \hat{t}_i | t_i), \forall t_i, \hat{t}_i \in T_i, \forall i \in N \\ & \sum_{t \in T} q(t) \sum_{i \in N} \chi_i(t) \leq && 0 \\ & \lambda_i(t_i) \geq 0, && \forall t_i \in T_i, \forall i \in N, \end{aligned}$$

For this linear program we can construct a Lagrangean function. We let $\alpha_i(\hat{t}_i | t_i)$ be the Lagrange multiplier for the constraint that says that type t_i should not hope to gain by reporting type \hat{t}_i to the state contingent contract being implemented by the mediator. With this we can write the Lagrangean of the linear programming problem as follows:

$$\begin{aligned} L(\mu, \chi, \lambda, \alpha) = & \sum_{i \in N} \sum_{t_i \in T_i} U_i(\mu, \chi | t_i) \\ & + \sum_{\hat{t}_i \in T_i} \alpha_i(\hat{t}_i | t_i) (U_i(\mu, \chi | t_i) - U_i(\mu, \chi, \hat{t}_i | t_i)) \end{aligned} \quad (8)$$

A. The Notion of Virtual Utility

We now introduce an important notion that was first developed by Myerson in [5]. This is the notion of virtual utility which is defined by a formula that takes incentive constraints into account. For any outcome $x \in X$, type profile $t \in T$, and any given vectors λ and α we define the virtual utility $v_i(x, t, \lambda, \alpha)$ for agent i as follows:

$$\begin{aligned} v_i(x, t, \lambda, \alpha) = & \frac{1}{q_i(t_i)} \left[(\lambda_i(t_i) + \sum_{\hat{t}_i \in T_i} \alpha_i(\hat{t}_i | t_i)) u_i(x, t) \right. \\ & \left. - \sum_{\hat{t}_i \in T_i} \alpha_i(t_i | \hat{t}_i) u_i(x, (t_{-i}, \hat{t}_i)) \right] \end{aligned} \quad (9)$$

Notice in equation (9) above that the Lagrange multiplier $\alpha_i(\hat{t}_i | t_i)$ is the dual variable corresponding to the incentive constraint which says an agent i of type t_i should not gain

by misrepresenting his type as \hat{t}_i . From linear programming theory, we know that this dual variable $\alpha_i(\hat{t}_i | t_i)$ will be non-zero when the corresponding constraint is tight. In the context of the MPNFI problem, if the constraint corresponding to the dual variable $\alpha_i(\hat{t}_i | t_i)$ is non-zero, then we can infer that agent i is tempted to misrepresent his type as \hat{t}_i when his actual type is t_i because he gets the same expected utility. From an inspection of equation (9), we can conclude that the virtual utility of agent i for an outcome $x \in X$ when the type profile is t magnifies the difference between the utilities of his true type t_i and the type \hat{t}_i that would tempt him to misrepresent.

Now, we can rewrite the Lagrangean in equation (8) by using equations (3), (4) and (9) as follows:

$$\begin{aligned} L(\mu, \chi, \lambda, \alpha) = & \sum_{t \in T} q(t) \sum_{x \in X} \mu(x | t) \sum_{i \in N} v_i(x, t, \lambda, \alpha) \\ & + \frac{1}{q_i(t_i)} \sum_{t \in T} q(t) \sum_{i \in N} \chi_i(t) \left[\sum_{\hat{t}_i \in T_i} \alpha_i(\hat{t}_i | t_i) \right. \\ & \left. - \sum_{\hat{t}_i \in T_i} \alpha_i(t_i | \hat{t}_i) + \lambda_i(t_i) \right] \end{aligned} \quad (10)$$

So, the linear programming problem given by equation (7) can be rewritten in terms of virtual utilities as follows:

$$\begin{aligned} \text{Max} & \sum_{t \in T} q(t) \sum_{x \in X} \mu(x | t) \sum_{i \in N} v_i(x, t, \lambda, \alpha) + \\ & \frac{1}{q_i(t_i)} \left(\sum_{t \in T} q(t) \sum_{i \in N} \chi_i(t) \left[\sum_{\hat{t}_i \in T_i} \alpha_i(\hat{t}_i | t_i) + \right. \right. \\ & \left. \left. - \sum_{\hat{t}_i \in T_i} \alpha_i(t_i | \hat{t}_i) + \lambda_i(t_i) \right] \right) \end{aligned} \quad (11)$$

s.t.

$$\sum_{t \in T} q(t) \sum_{i \in N} \chi_i(t) \leq 0 \quad (12)$$

$$\lambda_i(t_i) \geq 0, \forall t_i \in T_i, \forall i \in N, \text{ with at least one strict inequality} \quad (13)$$

B. Incentive Efficient Contracts

From this reformulation of the optimization problem, it is easy to note that the mediator must pick a state contingent contract that maximizes the sum of the virtual utilities of all types of all agents. Now, the Lagrangean in equation (10) can be maximized only if the coefficients of $\chi_i(t)$ are constant over all i and t . Such a constant can be set to 1 without loss of generality. A standard Lagrangean analysis now allows us to record the following proposition:

Proposition 4.2: A feasible state contingent contract (μ, χ) is incentive efficient if and only if there exist vectors λ and α such that:

$$\lambda_i(t_i) \geq 0 \text{ and } \alpha_i(\widehat{t}_i|t_i) \geq 0, \forall t_i \in T_i, \forall \widehat{t}_i \in T_i, \forall i \in N. \quad (14)$$

$$\lambda_i(t_i) + \sum_{\widehat{t}_i \in T_i} \alpha_i(\widehat{t}_i|t_i) - \sum_{\widehat{t}_i \in T_i} \alpha_i(t_i|\widehat{t}_i) = q_i(t_i). \quad (15)$$

$$\alpha_i(\widehat{t}_i|t_i) [U_i(\mu, \chi|t_i) - U_i(\mu, \chi, \widehat{t}_i|t_i)] = 0, \forall t_i, \widehat{t}_i \in T_i, \forall i \in N \quad (16)$$

$$\mu(x|t) > 0 \implies x \in \operatorname{argmax}_{y \in X} \sum_{i \in N} v_i(y, t, \lambda, \alpha), \forall t \in T, \forall x \in X. \quad (17)$$

Equation (14) comes from (a) our choice of the λ vector that is chosen to maximize the expected utilities and (b) the α vector corresponds to Lagrangean multipliers or dual variables corresponding to the incentive constraints, which by definition are non-negative. Equation (15) comes from setting the coefficients of $\chi_i(t)$ to unity. Equation (16) is nothing but the complementary slackness conditions corresponding to the incentive constraints of the original linear programming problem. Finally, equation (17) comes from the fact that the mediator is maximizing the sum of the virtual utilities of all the agents when he chooses the state contingent contract.

V. THE EQUITY PRINCIPLE

In the theory of the core for games with complete information, we are interested in establishing an allocation of surplus that inhibits agents from deviating and joining a coalition that can offer an alternative allocation of surplus which blocks the former. That is, we compare alternative contracts (allocations of surplus) with an established one. In extending this idea to the incomplete information case, we should think in terms of an established mediator who implements a state contingent contract that inhibits agents from deviating to cooperate with another blocking mediator who has a blocking state contingent contract to offer.

In addition, in the complete information case, we allow agents to compare an alternative contract with an established contract with the assumption that any agent who rejects the alternative will still get his allocation as specified by the established contract. For such an assumption to be workable, we then require that if any one agent rejects the alternative then all agents must continue to adhere to the established contract. That is, there can be no blocking without unanimity among all agents that are invited to block. In the incomplete information case, we must recognize the fact that some agents may be willing to block when they are of a certain type and not otherwise. So, an agent who agreed to block but was returned to the original contract would have now learnt new information which he could possibly use profitably in the established contract. So, to maintain the assumption that agents allocations as specified in the established contract are guaranteed, we will have to think about the blocking question being raised after the agents have sent in their type information to the established mediator, but before they are committed to

the state contingent contract that is to be implemented. This means that we need to formalize the blocking procedure and the blocking state contingent contracts.

A. Blocking Coalitions and Blocking State Contingent Contracts

The blocking procedure that we follow includes a blocking mediator. We assume that the blocking mediator may invite different coalitions according to some known randomized plan. The plan includes a specification of the probability of any coalition being chosen to implement a specific outcome that is feasible for that coalition. The outcome should of course depend on the information available to the coalition since it is unreasonable to allow blocking by a coalition to depend on information of agents outside the coalition.

So, we assume that a blocking mediator can ask any random subset S about their types and, based on their responses, must either invite all of S to join the blocking coalition to implement a jointly feasible outcome $x_S \in X_S$ or invite no coalition at all.

Such a blocking procedure may be characterized as follows: For any outcome $x_S \in X_S$ and any type profile $t_S \in T_S$ of the agents in S , we let $\nu_S(x_S|t_S)$ represent the probability that coalition S would be invited to block and implement the jointly feasible outcome $x_S \in X_S$ if the agents in S report a type profile t_S . Also, since we allowed the established mediator to make side-payments, we must also allow the blocking mediator to make side-payments. We do this by allowing the blocking mediator to specify the expected side-payment for each possible type of each agent. So, for each type t_i of each agent i , we let $\xi_i(t_i)$ be the blocking mediator's expected side-payment to agent i if i would be willing to block and report type t_i to the blocking mediator. With this we can define a blocking state contingent contract (ν, ξ) as follows:

Definition 5.1: A blocking state contingent contract by a blocking mediator is a pair of vectors (ν, ξ) such that:

- 1) $\nu = (\nu_S)_{S \subseteq N}$,
- 2) $\nu_S(x_S|t_S) \geq 0, \forall x \in X_S, \forall t_S \in T_S$,
- 3) $\sum_{S \subseteq N} \sum_{t_S \in T_S} \sum_{x_S \in X_S} \nu_S(x_S|t_S) \leq 1$, and
- 4) $\xi_i(t_i) \in \mathbb{R}, \forall t_i \in T_i, \forall i \in N$.

In definition (5.1) above, the blocking state contingent contract is a pair of vectors where the probability of any coalition S being picked to implement an outcome $x_S \in X_S$ when its type profile is t_S is always non-negative; and the probability of any such coalition being chosen by the blocking mediator is never greater than unity. With this definition of a blocking state contingent contract, we can now specify the blocking procedure.

1) *The Blocking Procedure:* We can describe the blocking procedure with this series of steps:

- First, according to the probability distribution specified by $\nu = (\nu_S)_{S \subseteq N}$, the blocking mediator chooses a random coalition $S \subseteq N$, a random profile of types $t_S \in T_S$ and a random outcome $x_S \in X_S$.
- The mediator then asks each of the agents in S whether he is willing to block and, if so, what his type is.

- If the agents in S all agree to block and their type profiles coincide exactly with t_S then the blocking mediator forms the coalition S and implements the jointly feasible outcome x_S .
- But if anyone of the agents does not agree to block or if the type profile does not match with t_S then he asks all the agents in S to continue with the state contingent contract from the established mediator.
- Now, when the blocking coalition does form, the planned monetary side-payments from the blocking mediator to the agents could depend on the blocking coalition S , the type profile t_S , and the jointly feasible outcome x_S that they implement. We let this be described by any function $\hat{\xi}(x_S, t_S)$ such that:

$$\sum_{S \ni \{i\}} \sum_{t_{S \setminus \{i\}} \in T_{S \setminus \{i\}}} \sum_{x_S \in X_S} \nu_S(x_S | t_S) \hat{\xi}_i(x_S, t_S) = \xi_i(t_i)$$

Now that we have formalized the notions of a blocking state contingent contract and the blocking procedure, to operationalize the idea of equity in selecting an implementable state contingent contract we would need to compare the utilities that such blocking state contingent contracts provide to agents in a blocking vis-a-vis the utilities that they derive by continuing to remain in the state contingent contract that an established mediator seeks to implement. We do this next.

2) *Tenable Blocking State Contingent Contracts*: For the purpose of comparing the welfare that agents get by either going along with a blocking mediator or staying with an established mediator, we let $\omega_i(t)$ denote the utility allocation from a state contingent contract of the established mediator that an agent i would lose when the type profile was t and he decided to join a blocking mediator. Let $\omega = (\omega_i(t))_{i \in N, t \in T}$ be a vector of such utility allocations. Given this vector of utility allocations, any blocking state contingent contract that a blocking mediator proposes must be such that it gives the agents more than what they can get in the established plan. We call such a state contingent contract a tenable state contingent contract and define it formally below:

Definition 5.2: A blocking state contingent contract (ν, ξ) is tenable against an established state contingent contract (μ, χ) which gives utility allocations $\omega = (\omega_i(t))_{t \in T, i \in N}$ if and only if it satisfies the conditions in equations 18 to 20:

Equation (18) states the fact that agents in a blocking coalition must not lose when they deviate from the established mediator; equation (19) is simply the incentive compatibility condition that says that agents must find it beneficial to report their true types to the blocking mediator when they have deviated from the established mediator; finally equation (20) simply says that the blocking mediator must get a non-negative payoff from forming a blocking coalition and implementing the blocking contract.

With this definition of a blocking state contingent contract, we can now sharpen our focus on isolating those contracts that are both efficient and equitable that an established mediator can hope to implement. Such contracts can be said to be inhibitive since they inhibit agents from cooperating with a blocking mediator and forming a blocking coalition that implements a blocking state contingent contract. We define

such contracts next.

B. Inhibitive State Contingent Contracts and Allocations

Recall from our discussion on the efficiency criterion in selecting a state contingent contract by an established mediator, we were able to define an optimization problem whose objective was to maximize the sum of virtual utilities of all the agents. That is, in selecting a state contingent contract to implement, the mediator would have to assume that the agents were behaving in a manner to maximize their virtual utilities and not their actual utilities. So, in order to operationalize the equity criteria in the selection of a contract, we would have to carry out the comparisons between contracts offered by an established mediator and a blocking mediator in virtual utility terms.

Recall now our utility allocation vector $\omega = (\omega_i(t))_{i \in N, t \in T}$. Such a utility allocation vector is said to be inhibitive if and only if there does not exist any blocking state contingent contract (ν, ψ) that is tenable against it. Since the Lagrangean function (11) that we are maximizing is specified in terms of virtual utilities, we would need to make these comparisons between the utility allocation vector ω and those from a blocking state contingent contract in virtual utility terms.

1) *A Virtual Utility Transformation of Inhibitive Allocations*: We let $\mathcal{V}_i(\omega, t, \lambda, \alpha)$ be the transformation of agent i 's utility allocations in ω into virtual utility in state t , according to the equation (9) with parameters λ and α . We therefore have the following relation:

$$\begin{aligned} \mathcal{V}_i(\omega, t, \lambda, \alpha) &= \frac{1}{q_i(t_i)} \left[(\lambda_i(t_i) + \sum_{\hat{t}_i \in T_i} \alpha_i(\hat{t}_i | t_i)) \omega_i(t) \right] \\ &\quad - \frac{1}{q_i(t_i)} \left[\sum_{\hat{t}_i \in T_i} \alpha_i(t_i | \hat{t}_i) \omega_i(t_{-i}, \hat{t}_i) \right] \end{aligned} \quad (21)$$

With this relation in place, we can now redefine an inhibitive allocation vector in terms of its virtual utilities. That is, we say that the utility allocation vector ω coming from a state contingent contract (μ, χ) is inhibitive if and only if there exist parameters λ and α such that, for any coalition S , the sum of virtual utilities that the members of S can expect with any outcome that is feasible for them, given all their information, is not more than the virtual-utility transformation of what they expect from the inhibitive utility allocation vector ω . We record this as a theorem below.

theorem 5.3: An allocation vector ω from a state contingent contract (μ, χ) offered for implementation by an established mediator is inhibitive if and only if there exist vectors λ and α such that:

- 1) $\lambda_i(t_i) + \sum_{\hat{t}_i \in T_i} \alpha_i(\hat{t}_i | t_i) - \sum_{\hat{t}_i \in T_i} \alpha_i(t_i | \hat{t}_i) = q_i(t_i), \forall t_i \in T_i, \forall i \in N,$
- 2) $\sum_{t_{N \setminus S} \in T_{N \setminus S}} q(t_{N \setminus S}) \sum_{i \in S} \mathcal{V}_i(\omega, t, \lambda, \alpha) \geq \sum_{t_{N \setminus S} \in T_{N \setminus S}} q(t_{N \setminus S}) \sum_{i \in S} v_i(x_S, t, \lambda, \alpha), \forall S \subseteq N, \forall x_S \in X_S, \forall t_S \in T_S,$
- 3) $\lambda_i(t_i) \geq 0$ and $\alpha_i(\hat{t}_i | t_i) \geq 0, \forall t_i \in T_i, \forall \hat{t}_i \in T_i, \forall i \in N.$

$$\xi_i(t_i) + \sum_{t_{-i} \in T_{-i}} q(t_{-i}) \sum_{S \supseteq \{i\}} \sum_{x_S \in X_S} \nu_S(x_S | t_S)(u_i(x_S, t) - \omega_i(t)) \geq 0, \forall t_i \in T_i, \forall i \in N. \quad (18)$$

$$\begin{aligned} & \xi_i(t_i) + \sum_{t_{-i} \in T_{-i}} q(t_{-i}) \sum_{S \supseteq \{i\}} \sum_{x_S \in X_S} \nu_S(x_S | t_S)(u_i(x_S, t) - \omega_i(t)) \\ & \geq \xi_i(\hat{t}_i) + \sum_{t_{-i} \in T_{-i}} q(t_{-i}) \sum_{S \supseteq \{i\}} \sum_{x_S \in X_S} \nu_S(x_S | t_{S-i}, \hat{t}_i)(u_i(x_S, t) - \omega_i(t)), \forall t_i \in T_i, \forall \hat{t}_i \in T_i, \forall i \in N. \end{aligned} \quad (19)$$

$$- \sum_{i \in N} \sum_{t_i \in T_i} q_i(t_i) \xi_i(t_i) \geq 0. \quad (20)$$

Proof: The proof for this theorem is along the lines of the proof in Theorem 1 in [1] and hence is omitted here. ■

The theorem basically says that the utility allocation vector is inhibitive if and only if there exist parameters λ and α such that, for any coalition S , the sum of all virtual utilities that the members of S can expect is not more than the sum of the virtual utility transformations of what they can expect from the ω given all their type information.

With this understanding of inhibitive allocations, we are ready to define the notion of the core as extended to cooperative games with incomplete information.

VI. THE INTERIM INCENTIVE COMPATIBLE FINE CORE OF THE MPNFI GAME

Recall from our preliminary discussion on the core for the MPNFI game in Section I that we assumed the mediator can make severance payments to agents who deviate from the established mediator to a blocking mediator. This assumption at first glance may seem surprising because such severance payments in the complete information case can never be beneficial. But in the incomplete information case they are serve an essential technical purpose in deriving the proof of existence of the core [1].

A. Balancedness and Balanced Games

For now, we let $\epsilon_i(t)$ denote the severance payment that agent i would get from the established mediator if he joined a blocking coalition after the type profile t was reported to the established mediator. We can now define the notion of an utility allocation vector that is *achievable* by a state contingent contract (μ, χ) .

Definition 6.1: A utility allocation vector $\omega = (\omega_i(t))_{t \in T, i \in N}$ is achievable by a state contingent contract (μ, χ) if and only if (μ, χ) is feasible (as defined in Definition 3.2) and there exists a promised vector of severance payments $\epsilon = (\epsilon_i(t))_{t \in T, i \in N}$ such that:

- 1) $\epsilon_i(t) \geq 0$, and
- 2) $\omega_i(t) = \chi_i(t) + \sum_{x \in X} \mu(x|t) u_i(x, t) - \epsilon_i(t), \forall t \in T, \forall i \in N.$

It is easy to see that $\omega_i(t)$ is the residual stake that an agent i has in the established plan which he stands to lose if he deviates to blocking coalition in state t . With this, we are ready to define the interim incentive compatible fine core of the MPNFI game and then examine its non-emptiness.

1) The Incentive Compatible Fine Core:

Definition 6.2: A utility allocation vector ω is said to be in the incentive compatible fine core if and only if ω is inhibitive and achievable by some feasible state contingent contract (μ, χ) .

In general, we have seen in the case of complete information games that (a) the non-emptiness of the core is not guaranteed and (b) to show non-emptiness of the core a balancedness condition should be satisfied. Our main result is to show that the incentive compatible fine core of the MPNFI game is non-empty. To show this we use the extension of the balancedness condition to incomplete information settings as introduced in [1].

2) Balancedness and Balancing Weights:

Definition 6.3: We let a vector of weights $\theta = (\theta_{S, x_S})_{x_S \in X_S, S \subseteq N}$ be a balanced collection of weights if and only if the following conditions are satisfied:

- 1) $\theta_{S, x_S} \geq 0 \forall x_S \in X_S, \forall S \subseteq N$
- 2) $\sum_{S \supseteq \{i\}} \sum_{x_S \in X_S} \theta_{S, x_S} = 1, \forall i \in N.$
- 3) **Balanced Games:**

Definition 6.4: We say that a game is balanced if for any balanced collection of weights $\theta = (\theta_{S, x_S})_{x_S \in X_S, S \subseteq N}$, there is some randomized strategy $\sigma \in \Delta(X)$ such that the following condition is satisfied.

$$\sum_{x \in X} \sigma(x) u_i(x, t) = \sum_{S \supseteq \{i\}} \sum_{x_S \in X_S} \theta_{S, x_S} u_i(x_S, t), \quad \forall t \in T, \forall i \in N. \quad (22)$$

Myerson [1] has shown that if the game is balanced then the core is non-empty. We record this as a theorem below which we use to show the non-emptiness of the incentive compatible fine core of the MPNFI game.

theorem 6.5: If a cooperative game with incomplete information is balanced then the incentive compatible fine core is non-empty.

B. Non-Emptiness of the Incentive Compatible Fine Core of the MPNFI Game

theorem 6.6: The incentive compatible fine core of the MPNFI game is non-empty.

Proof: To show that this theorem holds, we simply need to show that the MPNFI game is balanced. That is, we need

to show that there is some randomization over the set of outcomes ($\sigma \in \Delta(X)$) such that the condition given by equation (23) is satisfied for any balanced collection of weights $\theta = (\theta_{S,x_S})_{x_S \in X_S, S \subseteq N}$, for the class of MPNFI games.

$$\sum_{x \in X} \sigma(x) u_i(x, t) = \sum_{S \supseteq \{i\}} \sum_{x_S \in X_S} \theta_{S,x_S} u_i(x_S, t), \quad \forall t \in T, \forall i \in N \quad (23)$$

To show this we first consider two special cases of the balanced collection of weights $\theta = (\theta_{S,x_S})_{x_S \in X_S, S \subseteq N}$.

Case 1: Consider the collection of singleton subsets of N . With such a collection of subsets of N , it is easy to see that the only outcome possible for each subset is the no-trade outcome and we know that the utility that an agent gets from the no-trade outcome is zero. So, for any agent $i \in N$ and for any singleton coalition $S = \{i\}$, the set of outcomes is a singleton $\|X_S\| = 1$ and the utility of this outcome $x_S \in X_S$ is $u_i(x_S, t) = 0, \forall t \in T$. Such a collection of subsets can be a balanced collection if we associate the weights $\theta_{\{i\}} = 1$. Notice that we have dropped the subscript associated with the outcome since the set of outcomes is a singleton. With this set of balancing weights and utilities associated with the outcomes, it is easy to see that the LHS of equation (23) is always zero.

Now, looking at the RHS of equation (23), it is clear that we can always pick a randomization σ over the set of outcomes X such that the probability associated with the outcome which gives no agent any of the surplus is always 1. Such an outcome can be trivially constructed by giving all the surplus to the mediator. So, the RHS of (23) is also zero and we have a randomization over the set of outcomes such that the condition in equation (23) is satisfied.

Case 2: We now consider a balancing vector θ such that $\theta_{N,\hat{x}} = 1$ for some $\hat{x} \in X$ and $\theta_{S,x_S} = 0$ for all other $(S, x_S) \neq (N, \hat{x})$.

This can be easily proved and hence for reasons of conserving space is omitted.

The General Case: We now consider the case of an arbitrarily balanced collection, say

$\mathcal{C} = \{S_{1,x_{11}}, S_{1,x_{12}}, \dots, S_{1,x_{1k}}, S_{2,x_{21}}, \dots, S_{j,x_{j1}}, \dots, S_{l,x_{l1m}}\}$ where an element $S_{j,x_{j1}}$ of the set \mathcal{C} refers to the fact that coalition $S_j \subseteq N$ forms and implements the outcome $x_{j1} \in X_j$. Given the above balanced collection \mathcal{C} , from the definition of balancedness, we have the following relations.

$$\theta_{S,x_S} \geq 0, \forall x_S \in X_S, \forall S \in \mathcal{C} \quad (24)$$

$$\sum_{S \supseteq \{i\}; S \in \mathcal{C}} \sum_{x_S \in X_S} \theta_{S,x_S} = 1, \forall i \in N. \quad (25)$$

For this balanced collection, consider a partition of the set \mathcal{C} into two such that one of them includes all the elements $S_{j,x_{ji}}$ where the buying agent $(n+1) \in S$ and another where the buying agent is not included. We denote these sets as

$\mathcal{C}_{(n+1)}$ and $\mathcal{C}_{-(n+1)}$ respectively. Now consider the RHS of the balancedness condition given in equation 23:

$$\text{RHS} = \sum_{S \supseteq \{i\}} \sum_{x_S \in X_S} \theta_{S,x_S} u_i(x_S, t), \forall t \in T, \forall i \in N \quad (26)$$

Equation (26) can be rewritten by taking the summation over all coalitions in $S_{j,x_{ji}} \in \mathcal{C}_{(n+1)}$ and $S_{j,x_{ji}} \in \mathcal{C}_{-(n+1)}$.

$$\begin{aligned} \text{RHS} = & \sum_{S \in \mathcal{C}_{(n+1)}} \sum_{S \supseteq \{i\}} \sum_{x_S \in X_S} \theta_{S,x_S} u_i(x_S, t) + \\ & \sum_{S \in \mathcal{C}_{-(n+1)}} \sum_{S \supseteq \{i\}} \sum_{x_S \in X_S} \theta_{S,x_S} u_i(x_S, t), \\ & \forall t \in T, \forall i \in N \end{aligned} \quad (27)$$

From the structure of the MPNFI problem, it is clear that the only outcome possible for any coalition S that does not contain the buying agent is the no-trade outcome. The utility of the no-trade outcome is zero for all agents in a coalition S that does not contain the buying agent $(n+1)$. This means that in equation (27) above, we have $u_i(x_S, t) = 0, \forall i \in S, S \in \mathcal{C}_{-(n+1)}, \forall t \in T$. So, equation (27) can be written as

$$\text{RHS} = \sum_{S \in \mathcal{C}_{(n+1)}} \sum_{S \supseteq \{i\}} \sum_{x_S \in X_S} \theta_{S,x_S} u_i(x_S, t), \forall t \in T, \forall i \in N \quad (28)$$

Now, from the condition of the balanced collection of sets, we have:

$$\theta_{S,x_S} \geq 0, \forall S \in \mathcal{C}_{(n+1)} \quad (29)$$

$$\sum_{S \in \mathcal{C}_{(n+1)}} \sum_{x_S \in X_S} \theta_{S,x_S} = 1 \quad (30)$$

Equation (29) follows from equation (24). Equation (30) follows from the fact that the $\mathcal{C}_{(n+1)}$ contains all those sets which include the buying agent $(n+1)$ and sum of the weights associated with these sets and their outcomes must sum to unity given the fact that we are considering a balanced collection of weights.

This immediately implies the following:

$$\sum_{S \in \mathcal{C}_{(n+1)}} \sum_{S \ni i} \sum_{x_S \in X_S} \theta_{S,x_S} = 1, \text{ if } i = (n+1) \quad (31)$$

$$\sum_{S \in \mathcal{C}_{(n+1)}} \sum_{S \ni i} \sum_{x_S \in X_S} \theta_{S,x_S} \leq 1, \text{ if } i \neq (n+1) \quad (32)$$

So, the vector of balancing weights $(\theta_{S,x_S})_{x_S \in X_S, S \in \mathcal{C}_{(n+1)}}$ is akin to a probability distribution over the set of all possible outcomes $(X_S)_{S \in \mathcal{C}_{(n+1)}}$. Since any outcome that can be achieved by a coalition $S \subset N$ where $S \ni \{(n+1)\}$ can also be achieved by the grand coalition N , we can construct a randomization σ over the set of outcomes X such that the randomization simply assigns the same weight as the corresponding balancing weight to a particular outcome (S, x_S) .

With this, it is clear that given an arbitrary set of balancing weights, a randomization over the set of outcomes X is always possible such that the condition in Equation (23) always holds. This proves that the MPNFI game is balanced. And from Theorem 6.5 we can infer that the MPNFI game has a non-empty incentive compatible fine core. ■

VII. DISCUSSION AND CONCLUSION

In studying the procurement network formation problem when informational asymmetries exist, we have borrowed the conceptual apparatus from the stream of literature that extends the core to incomplete information settings [7], [8], [9], [10], [11]. Our result on the non-emptiness of the interim incentive compatible fine core of the multiple unit, single item procurement network formation problem shows clearly that a mediator, possibly a web based market maker can always come up with a mechanism to form the procurement network. The mechanism here is simply an implementation of the state contingent contract. However, before we can operationalize this there are several open issues that need to be addressed.

- 1) Our result on the non-emptiness of the incentive compatible fine core is a non-constructive existence result. We still need to develop an algorithmic procedure to identify a state contingent contract that is in the core of the game.
- 2) The interim incentive compatible core is an axiomatic exogenously imposed solution concept. If agents were to engage in endogenous non-cooperative play to agree upon a state contingent contract, then designing an extensive form game to reconcile the endogenous and exogenous viewpoints is an interesting question. Such games have been designed for the complete information setting, but we are not aware of any literature in the incomplete information context.

REFERENCES

- [1] R.B. Myerson, "Virtual utility and the core for games with incomplete information", Tech. Rep., Department of Economics, University of Chicago, Chicago, IL, 2005.
- [2] R. Wilson, "Information, efficiency and the core of an economy", *Econometrica*, vol. 46, no. 4, pp. 807–816, July 1978.
- [3] R.B. Myerson and M.A. Satterthwaite, "Efficient mechanisms for bilateral trading", *Journal of Economic Theory*, vol. 28, pp. 265–283, 1983.
- [4] R.B. Myerson, "Two-person bargaining problems with incomplete information", *Econometrica*, vol. 52, pp. 461–487, 1984.
- [5] R.B. Myerson, "Cooperative games with incomplete information", *International Journal of Game Theory*, vol. 13, pp. 69–96, 1984.
- [6] J.C Harsanyi, "Games with incomplete information played by Bayesian players", *Management Science*, vol. 14, no. 7, pp. 486–502, March 1968.
- [7] A.M. Kwasnica, "Bayesian implementable efficient and core allocations", 2000, Pennsylvania State University. Working Paper.
- [8] F. Forges, E. Minelli, and R. Vohra, "Incentives and the core of an exchange economy: A survey", *Journal of Mathematical Economics*, vol. 38, pp. 1–41, 2002.
- [9] D. Lee and O. Volij, "The core of economies with asymmetric information: An axiomatic approach", *Journal of Mathematical Economics*, vol. 38, pp. 43–63, 2002.
- [10] T. Ichiishi and A. Yamazaki, "Interim core concepts for a Bayesian pure exchange economy", *Journal of Mathematical Economics*, vol. 40, pp. 347–370, 2004.
- [11] G. de Clippel, "Values for cooperative games with incomplete information: An eloquent example", *Games and Economic Behavior*, vol. 53, pp. 73–82, 2005.

Tableau-based decision procedure for the full coalitional multiagent temporal-epistemic logic of branching time

Valentin Goranko

Dmitry Shkatov

Abstract

We develop a sound and complete tableau-based decision procedure for the full coalitional multiagent temporal-epistemic logic of branching time **CMATEL(CD+BT)** that extends logic **CTL** with epistemic operators for common and distributed knowledge for all coalitions of agents referred to in the language. The procedure runs in exponential time, which matches the lower bound established by Halpern and Vardi for a fragment of our logic, thus providing a complexity-optimal decision procedure and a complete deductive system for our logic.

1 Introduction

Reasoning about knowledge and time is crucial for designing, and verifying properties of, distributed and multiagent systems. A number of *temporal-epistemic logics* were proposed as logical frameworks for modeling of, and reasoning about, these aspects of distributed systems in the 1980's. This research was summarized in the comprehensive study by Halpern and Vardi [7]. In that study, the authors considered several essential characteristics of temporal-epistemic logics, namely: *one vs. several agents*, *synchrony vs. asynchrony*, *(no) learning*, *(no) forgetting*, *linear vs. branching time*, and the (non)existence of a *unique initial state*. Based on these, they identify and analyze 96 temporal-epistemic logics and obtain lower bounds for the complexity of the satisfiability problem in all of them. It turns out that most of the systems with more than one agents who do not learn or do not forget are undecidable, often Π_1^1 -hard (with common knowledge), or decidable but with non-elementary time lower bound (without common knowledge). For most of the remaining logics, the lower bounds established in [7] for the multiagent case range from PSPACE (synchronous systems without common knowledge), through EXPTIME (with common knowledge), to EXPSPACE (synchronous systems with no learning and unique initial state).

Despite the conceptual importance and wide range of po-

tential applications of temporal-epistemic logics, to the best of our knowledge, no efficient decision procedures for logics studied in [7] had been developed until quite recently, even for the systems with a relatively low known lower bounds. The only exception is [11], where a top-down tableau-style decision procedure for the logic **ATEL**, which subsumes the basic branching-time logic considered in [7] and this paper, was presented. In our view (to be explained further), however, [11] should be seen as a contribution to the complexity-theoretic analysis of the temporal-epistemic logics rather than to the development of efficient decision procedures for them.

In the recent precursor [5] to the present paper, we set out to fill in the above-mentioned gap by developing a practically efficient (within the theoretical complexity bounds) tableau-based decision procedure for the coalitional multiagent temporal-epistemic logic of linear time **CMATEL(CD+LT)** (for both the synchronous and asynchronous cases), building both on Wolper's incremental tableaux for **LTL** [13] and on our earlier work on tableaux for the full coalitional multiagent (purely) epistemic logic **CMAEL(CD)** [6].

In the present paper, we report on the second, and final, part of the project undertaken with the publication of [5]; namely, we present a sound, complete, and terminating incremental tableau for the *Coalitional Multi-Agent Temporal Epistemic Logic with operators for Common and Distributed knowledge and Branching Time*, **CMATEL(CD+BT)**. The tableau procedure presented herein follows the tableau-building philosophy developed for the logics **PDL** by Pratt in [9], **UB** by Ben-Ari, Manna and Pnueli in [1], and **CTL** by Emerson and Halpern in [3]. Our procedure essentially combines incremental tableaux for **CTL** from the latter (see also [10] for a recent detailed exposition) and tableaux for the full coalitional multiagent epistemic logic **CMAEL(CD)** developed in [6]. In the present paper, as in [6], we work with a more expressive epistemic language than the one considered in [7], as it contains operators for common and distributed knowledge for *all non-empty coalitions* (i.e., subsets) of the set of agents. The resulting decision procedure for testing satisfiability in **CMA-**

TEL(CD+BT) runs in exponential time, which is the optimal lower-bound, as established in [11].

We should mention that, even though the procedure presented in [11] can be used to test **CMATEL(CD+BT)**-formulae for satisfiability, this would not give us the optimal procedure, since such a procedure would *always* require exponential time predicted by the worst-case estimate. The incremental tableau presented in this paper, on the other hand, on average requires much less time than the theoretical upper bound (this claim cannot be made mathematically precise without an a priori probability distribution on formulae; however, it is substantiated by example in [4], where we compare the incremental tableaux presented in that paper with the top-down tableaux-style procedure from [12]).

Besides presenting the tableau-based procedure for **CMATEL(CD+BT)**, the other major objective of this paper is to demonstrate how two tableau procedures for logics with non-interacting fixed-point operators (the epistemic and the temporal ones, in our case) can be combined into a tableau procedure for the fusion of these logics, thus offering a contribution to the area of combination of logics (see e.g., [8]).

The present paper is structured as follows. In Section 2, we introduce the logic **CMATEL(CD+BT)**. In Section 3, we introduce Hintikka structures for **CMATEL(CD+BT)** and show that satisfiability of **CMATEL(CD+BT)**-formulae in Hintikka structures is equivalent to satisfiability in models introduced in Section 2. In Section 4, we present the tableau procedure for **CMATEL(CD+BT)** and, in Section 5, sketch out the proofs of soundness and completeness and briefly discuss the complexity of the procedure. The Appendix contains an example of a run of the procedure.

2 Syntax and semantics of the logic **CMATEL(CD+BT)**

The language \mathcal{L} of **CMATEL(CD+BT)** contains a (possibly infinite) set \mathbf{AP} of atomic propositions; the Boolean connectives \neg (“not”) and \wedge (“and”); the unary temporal operators $\exists\bigcirc$ and $\forall\bigcirc$ (existential and universal “next”, respectively); the binary temporal operators $\exists(-\mathcal{U}-)$ and $\forall(-\mathcal{U}-)$ (existential and universal “until”, respectively), as well as the unary epistemic operators $\mathbf{D}_A\varphi$ (“it is distributed knowledge among agents in A that φ ”), and $\mathbf{C}_A\varphi$ (“it is common knowledge among agents of A that φ ”) for every non-empty $A \subseteq \Sigma$, where Σ is the finite, non-empty set of names of agents belonging to \mathcal{L} . Subsets of Σ are called *coalitions*. Thus, the formulae of \mathcal{L} are defined as follows:

$$\begin{aligned} \varphi := & p \mid \neg\varphi \mid (\varphi_1 \wedge \varphi_2) \mid \exists\bigcirc\varphi \mid \forall\bigcirc\varphi \mid \\ & \mid \exists(\varphi_1 \mathcal{U}\varphi_2) \mid \forall(\varphi_1 \mathcal{U}\varphi_2) \mid \mathbf{D}_A\varphi \mid \mathbf{C}_A\varphi \end{aligned}$$

where p ranges over \mathbf{AP} and A ranges over the set $\mathcal{P}^+(\Sigma)$ of non-empty subsets of Σ . We write $\varphi \in \mathcal{L}$ to mean that φ is a formula of \mathcal{L} and $\Delta \subseteq \mathcal{L}$ to mean that Δ is a set of such formulae.

Thus, \mathcal{L} combines the language of Computational Tree Logic **CTL** [2] with the language of the full coalitional multi-agent epistemic logic **CMAEL(CD)** [6]. Although $\forall\bigcirc\varphi$ is definable as $\neg\exists\bigcirc\neg\varphi$, it is convenient to treat it as a primitive connective. The operator for individual knowledge $\mathbf{K}_a\varphi$ (“agent a knows that φ ”), where $a \in \Sigma$, can then be defined as $\mathbf{D}_{\{a\}}\varphi$, henceforth written $\mathbf{D}_a\varphi$. The other Boolean and temporal connectives can be defined as usual. We omit parentheses when this does not result in ambiguity.

Formulae of the form $\neg\mathbf{C}_A\varphi$ are *epistemic eventualities*, while those of the form $\exists(\varphi\mathcal{U}\psi)$ and $\forall(\varphi\mathcal{U}\psi)$ are *temporal eventualities*.

The semantics of temporal-epistemic logics considered in [7] is based on *system of runs* with m processors (agents). A run is a function from (the set of natural numbers) \mathbb{N} to the product L^m regarded as the set of *global states*, where L is the set of *local states*; each agent can be in one of local states at any moment in time. Thus, a global state is a tuple $\langle l_1, \dots, l_m \rangle$; the i -th component l_i of this global state representing the *local view* of the agent i . The pair (r, n) , where r is a run and $n \in \mathbb{N}$, is called in [7] a *point*. With every agent i , the authors of [7] associate the binary *epistemic indistinguishability relation* \sim_i on L^m , defined as follows: $\langle l_1, \dots, l_m \rangle \sim_i \langle l'_1, \dots, l'_m \rangle$ if $l_i = l'_i$; i.e., if the agent i has the same local views in these states.

According to [7], a system is *synchronous* when it has a ‘global clock’ observable by all agents and thus synchronizing their local times; formally, a system is synchronous if $(r, n) \sim_i (r', n')$ implies $n = n'$, for every $i = 1, \dots, m$, runs r, r' , and time moments n, n' . It turns out that the presence or absence of synchrony, under no other assumptions, does not affect the outcome of our tableau procedure, and therefore, the satisfiability of formulae.

The systems with (global) states represented as tuples of local states are generalized in [7] to systems where global states are abstract primitive entities and the epistemic relations are abstract equivalence relations on the set of such states. In the present paper, we work with this abstract semantics from [7]. We note that, as we show later, this semantics is *more general* than the above mentioned ‘concrete’ semantics from [7], despite the apparent assumption made in [7] that the two semantics are equivalent. We now turn to the presentation of the abstract semantics from [7].

Definition 2.1 A temporal-epistemic system (TES) is a tuple $\mathfrak{G} = (\Sigma, S, R, \{\mathcal{R}_A^D\}_{A \in \mathcal{P}^+(\Sigma)}, \{\mathcal{R}_A^C\}_{A \in \mathcal{P}^+(\Sigma)})$, where:

1. Σ is a finite, non-empty set of agents;
2. $S \neq \emptyset$ is a set of states;

3. $R \neq \emptyset$ is a set of runs: each $r \in R$ is a function $r : \mathbb{N} \mapsto S$. A state visited by a computation can, then, be represented as $r(n)$, where $r \in R$ and $n \in \mathbb{N}$. Also, with a state $r(n)$ we associate a pair (r, n) , referred to as a point; the set of all points in \mathfrak{G} is denoted by $P(\mathfrak{G})$. Notice that different points may be associated with the same state.
4. for every $A \in \mathcal{P}^+(\Sigma)$, \mathcal{R}_A^D and \mathcal{R}_A^C are binary relations on $P(\mathfrak{G})$, such that \mathcal{R}_A^C is the reflexive and transitive closure of $\bigcup_{A' \subseteq A} \mathcal{R}_{A'}^D$.

A TES \mathfrak{G} is synchronous (STES) if $((r, n), (r', n')) \in \mathcal{R}_A^D$ implies $n = n'$ for every $A \in \mathcal{P}^+(\Sigma)$.

Hereafter we write '(S)TES' to refer to general or synchronous temporal-epistemic system.

Definition 2.2 Let $(r, n) \in P(\mathfrak{G})$ for some (S)TES \mathfrak{G} with a set of runs R and let $r' \in R$. We say that r' extends (r, n) if $r(m) = r'(m)$ holds for all $m \leq n$.

Definition 2.3 A (synchronous) temporal-epistemic frame ((S)TEF) is a (S)TES $\mathfrak{G} = (\Sigma, S, R, \{\mathcal{R}_A^D\}_{A \in \mathcal{P}^+(\Sigma)}, \{\mathcal{R}_A^C\}_{A \in \mathcal{P}^+(\Sigma)})$, where each \mathcal{R}_A^D is an equivalence relation satisfying the following condition:

$$(\dagger) \quad \mathcal{R}_A^D = \bigcap_{a \in A} \mathcal{R}_{\{a\}}^D$$

If condition (\dagger) is replaced by the following, weaker one:

$$(\dagger\dagger) \quad \mathcal{R}_A^D \subseteq \mathcal{R}_B^D \text{ whenever } B \subseteq A,$$

then \mathfrak{F} is a (synchronous) temporal-epistemic pseudo-frame (pseudo-(S)TEF).

Notice that in (pseudo)-(S)TEFs \mathcal{R}_A^C is the transitive closure of $\bigcup_{a \in A} \mathcal{R}_{\{a\}}^D$, for every $A \in \mathcal{P}^+(\Sigma)$; furthermore, in such structures, each \mathcal{R}_A^C is an equivalence relation.

Definition 2.4 A (synchronous) temporal-epistemic model ((S)TEM, for short) is a tuple $\mathcal{M} = (\mathfrak{F}, L)$, where

- (i) \mathfrak{F} is a (S)TEF with a set of runs R ;
- (ii) $L : R \times \mathbb{N} \mapsto \mathcal{P}(\mathbf{AP})$ is a labeling function, such that $L(r, n)$ is the set of atomic propositions 'true' at a point (r, n) .

If condition (i) is changed so that \mathfrak{F} is a pseudo-(S)TEF, then \mathcal{M} is a (synchronous) temporal-epistemic pseudo-model (pseudo-(S)TEM).

Definition 2.5 The satisfaction relation \Vdash between (pseudo)-(S)TEMs, points, and formulae of \mathcal{L} is recursively defined as follows:

$$\mathcal{M}, (r, n) \Vdash p \text{ iff } p \in L(r, n);$$

$$\begin{aligned} \mathcal{M}, (r, n) \Vdash \neg\varphi & \text{ iff } \mathcal{M}, (r, n) \not\Vdash \varphi; \\ \mathcal{M}, (r, n) \Vdash \varphi \wedge \psi & \text{ iff } \mathcal{M}, (r, n) \Vdash \varphi \text{ and } \mathcal{M}, (r, n) \Vdash \psi; \\ \mathcal{M}, (r, n) \Vdash \exists\bigcirc\varphi & \text{ iff } \mathcal{M}, (r', n+1) \Vdash \varphi \text{ holds for some } \\ & r' \text{ extending } (r, n); \\ \mathcal{M}, (r, n) \Vdash \forall\bigcirc\varphi & \text{ iff } \mathcal{M}, (r', n+1) \Vdash \varphi \text{ holds for every } \\ & r' \text{ extending } (r, n); \\ \mathcal{M}, (r, n) \Vdash \exists(\varphi \mathcal{U} \psi) & \text{ iff, for some } r' \text{ extending } (r, n), \\ & \text{there exists } i \geq n \text{ such that } \mathcal{M}, (r', i) \Vdash \psi \text{ and} \\ & \mathcal{M}, (r', j) \Vdash \varphi \text{ holds for every } n \leq j < i; \\ \mathcal{M}, (r, n) \Vdash \forall(\varphi \mathcal{U} \psi) & \text{ iff, for every } r' \text{ extending } (r, n), \\ & \text{there exists } i \geq n \text{ such that } \mathcal{M}, (r', i) \Vdash \psi \text{ and} \\ & \mathcal{M}, (r', j) \Vdash \varphi \text{ holds for every } n \leq j < i; \\ \mathcal{M}, (r, n) \Vdash \mathbf{D}_A\varphi & \text{ iff } \mathcal{M}, (r', n') \Vdash \varphi \text{ for every} \\ & ((r, n), (r', n')) \in \mathcal{R}_A^D; \\ \mathcal{M}, (r, n) \Vdash \mathbf{C}_A\varphi & \text{ iff } \mathcal{M}, (r', n') \Vdash \varphi \text{ for every} \\ & ((r, n), (r', n')) \in \mathcal{R}_A^C. \end{aligned}$$

Satisfiability and validity of formulae are defined as usual.

Note that in the semantics above the labeling function L acts on points, not states, i.e., the semantics is point-based. To make the semantics state-based, one needs to impose the additional condition¹: $r(n) = r'(n')$ implies $L(r, n) = L(r', n')$. The two semantics differ: e.g., the formula $p \rightarrow \forall(\top \mathcal{U} p)$ is valid in the state-based semantics, but not in the point-based one.

The satisfaction condition for the operator \mathbf{C}_A can be paraphrased in terms of reachability. Let \mathfrak{F} be a (pseudo)-(S)TEF over the set of runs R and let $(r, n) \in R \times \mathbb{N}$. We say that point (r', n') is A -reachable from (r, n) if either $(r, n) = (r', n')$ or there exists a sequence $(r, n) = (r_0, n_0), (r_1, n_1), \dots, (r_{m-1}, n_{m-1}), (r_m, n_m) = (r', n')$ of points in $R \times \mathbb{N}$ such that, for every $0 \leq i < m$, there exists $a_i \in A$ such that $(r_i, n_i), (r_{i+1}, n_{i+1}) \in \mathcal{R}_{\{a_i\}}^D$. It is then easy to see that the following satisfaction condition for \mathbf{C}_A is equivalent to the one given above:

$$\mathcal{M}, (r, n) \Vdash \mathbf{C}_A\varphi \text{ iff } \mathcal{M}, (r', n') \Vdash \varphi \text{ for every } (r', n'), \\ A\text{-reachable from } (r, n).$$

Note that if $\Sigma = \{a\}$, then $\mathbf{D}_a\varphi \leftrightarrow \mathbf{C}_a\varphi$ is valid in every (S)TEM, for all $\varphi \in \mathcal{L}$. Thus, the single-agent case is essentially trivialized, so we assume throughout the rest of the paper that Σ contains at least 2 (names of) agents.

Also note that in models where states are tuples of local states, if $s \sim_i s'$ holds for every $i = 1, \dots, m$, then $s = s'$ and, therefore, the formula $p \rightarrow \mathbf{D}_\Sigma p$ is valid in every such model, but it is not valid in every (S)TEM. Thus, the abstract semantics presented above differs from the 'concrete' semantics presented in [7], despite the apparent assumption to the contrary made in [7].

Hereafter, we consider general temporal-epistemic systems; all definitions and results also apply to the synchronous variety, unless stated otherwise.

¹This condition is not imposed in [7], but this is an apparent omission because it is essentially assumed there.

3 Hintikka structures for CMA-TEL(CD+BT)

Even though we are ultimately interested in testing formulae of \mathcal{L} for satisfiability in a TEM, the tableau procedure we present tests for satisfiability in a more general kind of semantic structure—a *Hintikka structure*. We will show that $\theta \in \mathcal{L}$ is satisfiable in a TEM iff it is satisfiable in a Hintikka structure, hence the latter test is equivalent to the former. The advantage of working with Hintikka structures lies in the fact that they contain just as much semantic information about θ as is necessary for computing its truth value at a distinguished state. More precisely, while models provide the truth value of every formula of \mathcal{L} at every state, Hintikka structures only determine the truth values of formulae directly involved in the evaluation of a fixed formula θ , in the satisfiability of which we are interested. Another important difference between models and Hintikka structures is that, in Hintikka structures, the epistemic relations \mathcal{R}_A^D and \mathcal{R}_A^C only have to satisfy the properties laid down in Definition 2.1. All the other information about the desirable properties of epistemic relations is contained in the labeling of states in Hintikka structures. This labeling ensures that every Hintikka structure generates a pseudo-model (by the construction of Lemma 3.5), which can then be turned into a model.

Definition 3.1 A set $\Delta \subseteq \mathcal{L}$ is patently inconsistent if it contains a complementary pair of formulae (i.e., formulae ψ and $\neg\psi$ for some formula ψ).

A set $\Delta \subseteq \mathcal{L}$ is fully expanded if it is not patently inconsistent² and satisfies the following conditions, where $\text{Sub}(\psi)$ stands for the set of subformulae of a formula ψ :

1. if $\neg\neg\varphi \in \Delta$, then $\varphi \in \Delta$;
2. if $\varphi \wedge \psi \in \Delta$, then $\varphi \in \Delta$ and $\psi \in \Delta$;
3. if $\neg(\varphi \wedge \psi) \in \Delta$, then $\neg\varphi \in \Delta$ or $\neg\psi \in \Delta$;
4. if $\neg\exists\bigcirc\varphi \in \Delta$, then $\forall\bigcirc\neg\varphi \in \Delta$;
5. if $\neg\forall\bigcirc\varphi \in \Delta$, then $\exists\bigcirc\neg\varphi \in \Delta$;
6. if $\exists(\varphi\mathcal{U}\psi) \in \Delta$, then $\psi \in \Delta$ or $\varphi, \exists\bigcirc\exists(\varphi\mathcal{U}\psi) \in \Delta$;
7. if $\neg\exists(\varphi\mathcal{U}\psi) \in \Delta$, then $\neg\psi, \neg\varphi \in \Delta$ or $\neg\psi, \neg\exists\bigcirc\exists(\varphi\mathcal{U}\psi) \in \Delta$;
8. if $\forall(\varphi\mathcal{U}\psi) \in \Delta$, then $\psi \in \Delta$ or $\varphi, \forall\bigcirc\forall(\varphi\mathcal{U}\psi) \in \Delta$;
9. if $\neg\forall(\varphi\mathcal{U}\psi) \in \Delta$, then $\neg\psi, \neg\varphi \in \Delta$ or $\neg\psi, \neg\forall\bigcirc\forall(\varphi\mathcal{U}\psi) \in \Delta$;
10. if $\mathbf{D}_A\varphi \in \Delta$, then $\mathbf{D}_{A'}\varphi \in \Delta$ for every A' such that $A \subseteq A' \subseteq \Sigma$;
11. if $\mathbf{D}_A\varphi \in \Delta$, then $\varphi \in \Delta$;
12. if $\mathbf{C}_A\varphi \in \Delta$, then $\mathbf{D}_a(\varphi \wedge \mathbf{C}_A\varphi) \in \Delta$ for every $a \in A$;

²Even though in general, not being patently inconsistent is a weaker condition than a propositional consistency, in the case of fully expanded sets, they coincide.

13. if $\neg\mathbf{C}_A\varphi \in \Delta$, then $\neg\mathbf{D}_a(\varphi \wedge \mathbf{C}_A\varphi) \in \Delta$ for some $a \in A$;
14. if $\psi \in \Delta$ and $\mathbf{D}_A\varphi \in \text{Sub}(\psi)$, then either $\mathbf{D}_A\varphi \in \Delta$ or $\neg\mathbf{D}_A\varphi \in \Delta$.

Definition 3.2 A temporal-epistemic Hintikka structure (TEHS) is a tuple $(\Sigma, S, R, \{\mathcal{R}_A^D\}_{A \in \mathcal{P}^+(\Sigma)}, \{\mathcal{R}_A^C\}_{A \in \mathcal{P}^+(\Sigma)}, H)$ such that $(\Sigma, S, R, \{\mathcal{R}_A^D\}_{A \in \mathcal{P}^+(\Sigma)}, \{\mathcal{R}_A^C\}_{A \in \mathcal{P}^+(\Sigma)})$ is a TES, and H is a labeling of points $(r, n) \in R \times \mathbb{N}$ with sets of formulae of \mathcal{L} satisfying the following constraints, for all (r, n) :

- H1** $H(r, n)$ is fully expanded;
- H2** if $\neg\varphi \in H(r, n)$, then $\varphi \notin H(r, n)$;
- H3** if $\exists\bigcirc\varphi \in H(r, n)$, then $\varphi \in H(r', n+1)$ holds for some r' extending (r, n) ;
- H4** if $\forall\bigcirc\varphi \in H(r, n)$, then $\varphi \in H(r', n+1)$ holds for every r' extending (r, n) ;
- H5** if $\exists(\varphi\mathcal{U}\psi) \in H(r, n)$, then, for some r' extending (r, n) , there exists $i \geq n$ such that $\psi \in H(r', i)$ and $\varphi \in H(r', j)$ holds for every $n \leq j < i$;
- H6** if $\forall(\varphi\mathcal{U}\psi) \in H(r, n)$, then, for every r' extending (r, n) , there exists $i \geq n$ such that $\psi \in H(r', i)$ and $\varphi \in H(r', j)$ holds for every $n \leq j < i$;
- H7** if $\neg\mathbf{D}_A\varphi \in H(r, n)$, then there exists $r' \in R$ and $n' \in \mathbb{N}$ such that $((r, n), (r', n')) \in \mathcal{R}_A^D$ and $\neg\varphi \in H(r', n')$;
- H8** if $((r, n), (r', n')) \in \mathcal{R}_A^D$, then $\mathbf{D}_{A'}\varphi \in H(r, n)$ iff $\mathbf{D}_{A'}\varphi \in H(r', n')$ holds for every $A' \subseteq A$;
- H9** if $\neg\mathbf{C}_A\varphi \in H(r, n)$, then there exists $r' \in R$ and $n' \in \mathbb{N}$ such that $((r, n), (r', n')) \in \mathcal{R}_A^C$ and $\neg\varphi \in H(r', n')$.

Synchronous temporal-epistemic Hintikka structures are defined accordingly.

Definition 3.3 A formula θ is satisfiable in a TEHS \mathcal{H} with a labeling function H if $\theta \in H(r, n)$ for some point (r, n) of \mathcal{H} . A set of formulae Θ is satisfiable in \mathcal{H} if $\Theta \subseteq H(r, n)$ for some point (r, n) of \mathcal{H} .

Now, we show that $\theta \in \mathcal{L}$ is satisfiable in a TEM iff it is satisfiable in a TEHS. One direction is almost immediate, as every TEM naturally induces a TEHS. More precisely, given a TEM \mathcal{M} , we define the *extended labeling* $L_{\mathcal{M}}^+$ on the set of points of \mathcal{M} as follows: $L_{\mathcal{M}}^+(r, n) = \{\varphi \mid \mathcal{M}, (r, n) \Vdash \varphi\}$ for every (r, n) . The following claim is then straightforward.

Lemma 3.4 Let $\mathcal{M} = (\Sigma, S, R, \{\mathcal{R}_A^D\}_{A \in \mathcal{P}^+(\Sigma)}, \{\mathcal{R}_A^C\}_{A \in \mathcal{P}^+(\Sigma)}, L)$ be a TEM satisfying $\theta \in \mathcal{L}$, and let $L_{\mathcal{M}}^+$ be the extended labeling on \mathcal{M} . Then, $\mathcal{H} = (\Sigma, S, R, \{\mathcal{R}_A^D\}_{A \in \mathcal{P}^+(\Sigma)}, \{\mathcal{R}_A^C\}_{A \in \mathcal{P}^+(\Sigma)}, L^+)$ is a TEHS satisfying θ .

To establish the converse, we first prove that the existence of a Hintikka structure satisfying θ implies the existence of a pseudo-model satisfying θ ; then, we prove that this in turn implies the existence of a model satisfying θ .

Lemma 3.5 *If $\theta \in \mathcal{L}$ is satisfiable in a TEHS, then it is satisfiable in a pseudo-TEM.*

Proof. Let $\mathcal{H} = (\Sigma, S, R, \{\mathcal{R}_A^D\}_{A \in \mathcal{P}^+(\Sigma)}, \{\mathcal{R}_A^C\}_{A \in \mathcal{P}^+(\Sigma)}, H)$ be a TEHS satisfying θ . We build a pseudo-TEM satisfying θ as follows. First, for every $A \in \mathcal{P}^+(\Sigma)$, let \mathcal{R}'_A be the reflexive, symmetric, and transitive closure of $\bigcup_{A \subseteq B} \mathcal{R}_B^D$ and let \mathcal{R}^C_A be the transitive closure of $\bigcup_{a \in A} \mathcal{R}'_a$. Notice that $\mathcal{R}_A^D \subseteq \mathcal{R}'_A$ and $\mathcal{R}_A^C \subseteq \mathcal{R}^C_A$ for every $A \in \mathcal{P}^+(\Sigma)$. Next, let $L(r, n) = H(r, n) \cap \mathbf{AP}$, for every point $(r, n) \in R \times \mathbb{N}$. It is then easy to check that $\mathcal{M}' = (\Sigma, S, R, \{\mathcal{R}'_A\}_{A \in \mathcal{P}^+(\Sigma)}, \{\mathcal{R}^C_A\}_{A \in \mathcal{P}^+(\Sigma)}, L)$ is a pseudo-TEM.

To complete the proof of the lemma, we show, by induction on the formula $\chi \in \mathcal{L}$ that, for every point (r, n) and every $\chi \in \mathcal{L}$, the following hold:

- (i) $\chi \in H(r, n)$ implies $\mathcal{M}', (r, n) \Vdash \chi$;
- (ii) $\neg\chi \in H(r, n)$ implies $\mathcal{M}', (r, n) \Vdash \neg\chi$.

Let χ be some $p \in \mathbf{AP}$. Then, $p \in H(r, n)$ implies $p \in L(r, n)$ and thus, $\mathcal{M}', (r, n) \Vdash p$; if, on the other hand, $\neg p \in H(r, n)$, then due to (H2), $p \notin L(r, n)$ and thus $p \notin L(r, n)$; hence, $\mathcal{M}', (r, n) \Vdash \neg p$.

Assume that the claim holds for all subformulae of χ ; then, we have to prove that it holds for χ , as well.

Suppose that $\chi = \neg\varphi$. If $\neg\varphi \in H(r, n)$, then the inductive hypothesis immediately gives us $\mathcal{M}', (r, n) \Vdash \neg\varphi$; if $\neg\neg\varphi \in H(r, n)$, then by virtue of (H1), $\varphi \in H(r, n)$ and hence, by inductive hypothesis, $\mathcal{M}', (r, n) \Vdash \varphi$ and thus $\mathcal{M}', (r, n) \Vdash \neg\neg\varphi$.

The cases of $\chi = \varphi \wedge \psi$, $\chi = \exists\bigcirc\varphi$, and $\chi = \forall\bigcirc\varphi$ are straightforward, using (H1) – (H4).

Let χ be $\exists(\varphi\mathcal{U}\psi)$. If $\exists(\varphi\mathcal{U}\psi) \in H(r, n)$, then the desired conclusion immediately follows from (H5) and the inductive hypothesis. If $\neg\exists(\varphi\mathcal{U}\psi) \in H(r, n)$, then due to (H1), either $\neg\psi, \neg\varphi \in H(r, n)$ or $\neg\psi, \neg\exists\bigcirc\exists(\varphi\mathcal{U}\psi) \in H(r, n)$. In the former case, the conclusion immediately follows from the inductive hypothesis. Otherwise, due to (H1) and (H4), $\neg\exists(\varphi\mathcal{U}\psi) \in H(r', n+1)$ holds for every run r' extending (r, n) . By repeating the argument, we obtain that, for every run r' extending (r, n) , either $\neg\varphi \in H(r', i)$ for some $i \geq 0$ and $\neg\psi \in H(r', j)$ for every $0 \leq j \leq i$ or $\neg\psi \in H(r', i)$ for every $i \geq 0$. In either case, the inductive hypothesis implies that $\mathcal{M}, (r, n) \Vdash \neg\exists(\varphi\mathcal{U}\psi)$, as desired.

The case of $\chi = \forall(\varphi\mathcal{U}\psi)$ is similar to the previous one and is left to the reader.

Suppose that $\chi = \mathbf{D}_A\varphi$. Assume, first, that $\mathbf{D}_A\varphi \in H(r, n)$. In view of the inductive hypothesis, it suffices to show that $((r, n), (r', n')) \in \mathcal{R}'_A$ implies $\varphi \in H(r', n')$.

So, assume that $((r, n), (r', n')) \in \mathcal{R}'_A$. There are two cases to consider. If $(r, n) = (r', n')$, then the conclusion immediately follows from (H1). Otherwise, there exists an undirected path from (r, n) to (r', n') along the relations \mathcal{R}'_A , where each A' is a superset of A . Then, due to (H8), $\mathbf{D}_A\varphi \in H(r', n')$; hence, by (H1), $\varphi \in H(r', n')$, as desired.

Now, let $\neg\mathbf{D}_A\varphi \in H(r, n)$. In view of the inductive hypothesis, it suffices to show that there exist $r' \in R$ and $n' \in \mathbb{N}$ such that $((r, n), (r', n')) \in \mathcal{R}'_A$ and $\neg\varphi \in H(r', n')$. By (H7), there exists $r' \in R$ and $n' \in \mathbb{N}$ such that $((r, n), (r', n')) \in \mathcal{R}_A^D$ and $\neg\varphi \in H(r', n')$. As $\mathcal{R}_A^D \subseteq \mathcal{R}'_A$, the desired conclusion follows.

Suppose that $\chi = \mathbf{C}_A\varphi$. Assume that $\mathbf{C}_A\varphi \in H(r, n)$. In view of the inductive hypothesis, it suffices to show that if (r', n') is A -reachable from (r, n) in \mathcal{M}' , then $\varphi \in H(r', n')$. If $(r, n) = (r', n')$ the claim follows from (H1). So, suppose that, for some $m \geq 1$, there exists a sequence of points $(r, n) = (r_0, n_0), \dots, (r_{m-1}, n_{m-1}), (r_m, n_m) = (r', n')$ such that, for every $0 \leq i < m$, there exists $a_i \in A$ such that $((r_i, n_i), (r_{i+1}, n_{i+1})) \in \mathcal{R}'_{a_i}$. Then, for every $0 \leq i < m$, there exists a path from (r_i, n_i) to (r_{i+1}, n_{i+1}) along relations $\mathcal{R}'_{A'}$ such that $a_i \in A'$ for every A' . Then, we can show by induction on i , using (H1) and (H8), that $\mathbf{C}_A\varphi \in H(r_i, n_i)$ holds for every $0 \leq i < m$. Indeed, this holds for $i = 0$; assuming that it holds for some i , by (H1)(12) we have that $\mathbf{D}_{a_i}(\varphi \wedge \mathbf{C}_A\varphi) \in H(r_i, n_i)$, hence, by (H1)(10) and (H8), $\varphi \in H(r_{i+1}, n_{i+1})$. Now, by taking $i = m - 1$ we obtain $\varphi \in H(r', n')$, as required.

Finally, assume that $\neg\mathbf{C}_A\varphi \in H(r, n)$. Then, the desired conclusion follows from (H9), the fact that $\mathcal{R}_A^C \subseteq \mathcal{R}'_A$, and the inductive hypothesis. \square

Lemma 3.6 *If $\theta \in \mathcal{L}$ is satisfiable in a pseudo-TEM, then it is satisfiable in a TEM.*

Proof. The proof is exactly the same as in [5, Section 3], as the pseudo-models are only ‘defective’ with respect to epistemic, but not temporal, relations; therefore, the construction for branching time is the same as for linear time. \square

Lemmas 3.4, 3.5, and 3.6 immediately give us the following theorem.

Theorem 3.7 *A $\theta \in \mathcal{L}$ is satisfiable in a TEM iff it is satisfiable in a TEHS.*

4 Tableau procedure for CMATEL(CD+BT)

In this section, we present a tableau procedure for **CMA-TEL(CD+BT)**. We describe a procedure for testing for satisfiability in synchronous models, as it requires extra care. We then briefly mention how the general case is different

and argue that the outcome of the procedure is the same in both cases, implying that, satisfiability-wise, general and synchronous semantics are equivalent.

4.1 Overview of the procedure

The tableau procedure for testing a formula $\theta \in \mathcal{L}$ for satisfiability attempts to construct a non-empty graph \mathcal{T}^θ (called *tableau*), whose nodes are finite sets of \mathcal{L} -formulae, encoding ‘sufficiently many’ TEHSs for θ , in the sense that if θ is satisfiable, then it is satisfiable in a TEHS represented by \mathcal{T}^θ . The main ideas underlying our tableau algorithm come from the tableau procedures for the logics **PDL** in [9], **UB** in [1] and **CTL** in [3] (see also a detailed exposition of tableaux for **CTL** in [10]), as well as recently developed tableaux for multiagent epistemic logics in [6]. To make the present paper self-contained, we outline the basic ideas behind our tableau algorithm in line with those references, and then describe the particulars specific to **CMA-TEL(CD+BT)**.

Usually, tableaux work by decomposing the input formula into simpler formulae, in accordance with the semantics of the logical connectives. In the classical propositional case, “simpler” implies shorter, thus ensuring the termination of the procedure. The decomposition into simpler formulae in the tableau for classical propositional logic produces a tree representing an exhaustive search for a Hintikka set, the classical propositional analogue of Hintikka structures, for the input formula θ . If at least one leaf of that tree is a Hintikka set for θ , the search has succeeded and θ is proved satisfiable; otherwise, it is declared unsatisfiable.

When applied to logics containing fixpoint-definable operators, such as \mathbf{C}_A , $\exists \mathcal{U}$, and $\forall \mathcal{U}$, these two defining features of the classical tableau method no longer apply. First, the decomposition of the fixpoint formulae, which is done by unfolding their fixpoint definitions, usually produces larger formulae: $\mathbf{C}_A\varphi$ is decomposed into the formulae $\mathbf{D}_a(\varphi \wedge \mathbf{C}_A\varphi)$; analogously for formulae of the form $\exists(\varphi \mathcal{U}\psi)$ and $\forall(\varphi \mathcal{U}\psi)$. Hence, we need a termination-ensuring mechanism. In our tableaux, such a mechanism is provided by the use (and reuse) of so called “pre-states”, whose role is to ensure the finiteness of the construction and, hence, termination of the procedure. Second, the only reason why a tableau may fail to produce a Hintikka set for the input formula in the classical case is that every attempt to build such a set results in a collection of formulae containing a *patent inconsistency*, i.e., a complementary pair of formulae $\varphi, \neg\varphi$. In the case of **CMA-TEL(CD+BT)**, there are other such reasons, specific to TEHS, which are more involved structures than classical Hintikka sets. One such reason has to do with eventualities: the truth of an eventuality at a state s in a TEM \mathcal{M} requires existence of a path going from s to a state of \mathcal{M} at which the ‘promise’ of that

eventuality is fulfilled. Since truth in TEMs is simulated by membership in state labels of Hintikka structures, eventualities impose respective conditions on the labels. Thus, the presence of an eventuality $\neg\mathbf{C}_A\varphi$ in the label of a state s of a TEHS \mathcal{H} requires the existence in \mathcal{H} of an A -path (i.e. a path along relations of the form \mathcal{R}_B^D , where $\mathcal{R}_B^D \subseteq \mathcal{R}_A^D$) from s to a state t whose label contains $\neg\varphi$, due to condition (H9) of Definition 3.2. Similar requirements apply to eventualities of the form $\exists(\varphi \mathcal{U}\psi)$ and $\forall(\varphi \mathcal{U}\psi)$ due to conditions (H5) and (H6) of Definition 3.2. The tableau analogs of these conditions are called *realization of eventualities*. If a tableau contains a node with an unrealized eventuality in its label, then it cannot produce a TEHS, and thus is ‘bad’ and needs repairing by removing such nodes. The third possible reason for a tableau to be ‘bad’ has to do with successor nodes: it may so happen that some of the required successors of a node s are missing from the tableau; then, s is ‘bad’, and hence needs to be removed. Notice that in TEHSs, and thus in tableaux, states have two kinds of successors: temporal and epistemic. The absence of either kind of successor can ruin the chances of a tableau node to correspond to a state of a TEHS.

The tableau procedure consists of three major phases: *pretableau construction*, *pre-state elimination*, and *state elimination*. During the first, we produce the *pretableau* for θ —a directed graph \mathcal{P}^θ , from which the tableau \mathcal{T}^θ will be extracted. The nodes of \mathcal{P}^θ are sets of formulae coming in two varieties: *states* and *pre-states*. States are fully expanded sets, meant to represent (labels of) states of a Hintikka structure, while pre-states only play a temporary role in the construction of \mathcal{T}^θ . During the second phase, all pre-states from \mathcal{P}^θ are removed and their incoming edges are redirected, creating a smaller graph \mathcal{T}_0^θ , the *initial tableau* for θ . Finally, we remove from \mathcal{T}_0^θ all states, if any, that cannot be satisfied in a TEHS, for any of the reasons mentioned above. The elimination procedure results in a (possibly empty) subgraph \mathcal{T}^θ of \mathcal{T}_0^θ , called the *final tableau* for θ . If some state Δ of \mathcal{T}^θ contains θ , we declare θ satisfiable; otherwise, we declare it unsatisfiable. An example illustrating the tableau construction is provided in Appendix A.

4.2 Pretableau construction phase

All states and pre-states of the pretableau \mathcal{P}^θ constructed during this phase are ‘time-stamped’. Whenever necessary to make it explicit, we will use the notation $\Gamma^{[k]}$ indicating that pre-state Γ was created as the k th component of a run; likewise for states.

The pretableau contains three types of edge, described below. As already mentioned, a procedure attempts to produce a compact representation of a sufficiently many TEHSs for the input formula θ by organizing an exhaustive search for such structures. One type of edge, depicted by

unmarked double arrows \Longrightarrow , represents the search transitions in the tableau. The exhaustive search considers all possible alternatives that arise when expanding pre-states into states through branching when dealing with disjunctive formulae. Thus, when we draw a double arrow from a pre-state Γ to states Δ and Δ' (depicted as $\Gamma \Longrightarrow \Delta$ and $\Gamma \Longrightarrow \Delta'$, respectively), this intuitively means that, in any TEHS, a state whose label extends the set Γ has to contain at least one of Δ and Δ' . Our first construction rule, **(SR)**, prescribes how to create tableau states from pre-states.

Given a set $\Gamma \subseteq \mathcal{L}$, we say that Δ is a *minimal, fully expanded extension* of Γ if Δ is fully expanded, $\Gamma \subseteq \Delta$, and there is no Δ' such that $\Gamma \subseteq \Delta' \subset \Delta$ and Δ' is fully expanded.

Rule (SR) Given a pre-state $\Gamma^{[k]}$ such that **(SR)** has not been applied to $\Gamma^{[k]}$ earlier, do the following:

1. Add to the pretableau all minimal fully expanded extensions $\Delta^{[k]}$ of $\Gamma^{[k]}$ as *states*;
2. if $\Delta^{[k]}$ contains no formulae of the form $\exists \circ \varphi$, add $\exists \circ \top$ to it;
3. for each so obtained state $\Delta^{[k]}$, put $\Gamma^{[k]} \Longrightarrow \Delta^{[k]}$;
4. if, however, the pretableau already contains a state $\Delta'^{[m]}$ that coincides with $\Delta^{[k]}$, do not create another copy of $\Delta'^{[m]}$, but only put $\Gamma^{[k]} \Longrightarrow \Delta'^{[m]}$.

We denote by $\text{states}(\Gamma)$ the (finite) set of states $\{\Delta \mid \Gamma \Longrightarrow \Delta\}$.

Notice that in all construction rules, as in **(SR)**, we allow reuse of (pre)states, which were originally stamped with a possibly different time-stamp. This does not correspond to one (pre)state being part of two different runs at different moments of time; rather, the ‘futures’ of these runs, starting from the reused (pre)state, can be assumed to be identical, modulo the time difference.

The second type of edge in a pretableau represents epistemic relations in the TEHS that the procedure attempts to build. This type of edge is represented by single arrows marked with epistemic formulae whose presence in the source state requires the presence in the tableau of a target state, reachable by a particular epistemic relation. All such formulae have the form $\neg \mathbf{D}_A \varphi$, as can be seen from Definition 3.2. Intuitively if, say $\neg \mathbf{D}_A \varphi \in \Delta^{[k]}$, then we need some pre-state $\Gamma^{[k]}$ containing $\neg \varphi$ to be accessible from $\Delta^{[k]}$ by \mathcal{R}_A^D .³ The reason we mark these single arrows by a formula $\neg \mathbf{D}_A \varphi$ (rather than by just coalition A), is that we have to remember why we had to create this particular Γ , and not just what relation connects Δ to Γ . This information will be needed during the elimination phases. We now formulate the rule producing this second type of edge.

³We require the newly created pre-states to bear the same time stamp as the source state for the sake of synchrony, as this reflects the fact that all epistemic alternatives belong to the same temporal level of any TEHS.

Rule (DR): Given a state $\Delta^{[k]}$ such that $\neg \mathbf{D}_A \varphi \in \Delta^{[k]}$ and **(DR)** has not been applied to $\Delta^{[k]}$ earlier, do the following:

1. Create a new pre-state $\Gamma^{[k]} = \{\neg \varphi\} \cup \bigcup_{A' \subseteq A} \{\mathbf{D}_{A'} \psi \mid \mathbf{D}_{A'} \psi \in \Delta\} \cup \bigcup_{A' \subseteq A} \{\neg \mathbf{D}_{A'} \psi \mid \neg \mathbf{D}_{A'} \psi \in \Delta\}$.
2. If pre-state $\Gamma^{[k]}$ is patently inconsistent, remove it.
3. Otherwise, connect $\Delta^{[k]}$ to $\Gamma^{[k]}$ with $\xrightarrow{\neg \mathbf{D}_A \varphi}$.
4. If, however, the tableau already contains a pre-state $\Gamma'^{[k]} = \Gamma^{[k]}$, do not add another copy of $\Gamma'^{[k]}$, but simply connect $\Delta^{[k]}$ to $\Gamma'^{[k]}$ with $\xrightarrow{\neg \mathbf{D}_A \varphi}$.

The third type of edge, depicted by single arrows marked with formulae of the form $\exists \circ \varphi$, represent temporal transitions in TEHSs that the tableau is trying to build. The rationale for this rule is similar to that for **(DR)**, the only difference being that we are now considering temporal, rather than epistemic, formulae forcing creation of new pre-states.

Rule (Next): Given a state $\Delta^{[k]}$ such that **(Next)** has not been applied to $\Delta^{[k]}$ earlier, do the following:

1. For each $\exists \circ \varphi \in \Delta^{[k]}$, create a new pre-state $\Gamma^{[k+1]} = \{\varphi\} \cup \{\psi \mid \forall \circ \psi \in \Delta^{[k]}\}$.⁴
2. If pre-state $\Gamma^{[k]}$ is patently inconsistent, remove it immediately.
3. Otherwise, connect $\Delta^{[k]}$ to $\Gamma^{[k+1]}$ with $\xrightarrow{\exists \circ \varphi}$.
4. If, however, the tableau already contains a pre-state $\Gamma'^{[m]} = \Gamma^{[k+1]}$, do not add another copy of $\Gamma'^{[m]}$, but simply connect $\Delta^{[k]}$ to $\Gamma'^{[m]}$ with $\xrightarrow{\exists \circ \varphi}$.

We now describe the order of application of the above rules. We start off by creating a single pre-state $\{\theta\}$, containing the input formula. Then, we alternately apply **(DR)** and **(Next)** to the states created at the previous stage and then applying **(SR)** to the newly created pre-states. The construction stage is over when the applications of **(DR)** and **(Next)** do not produce any new pre-states.

4.3 Pre-State elimination phase

At this phase we remove from \mathcal{P}^θ all pre-states and double arrows, which results in a smaller graph \mathcal{T}_0^θ called the *initial tableau*. Formally, we apply the following rule:

Rule (PR) For every pre-state Γ in \mathcal{P}^θ , do the following:

1. Remove Γ from \mathcal{P}^θ .
2. If there is a state Δ in \mathcal{P}^θ with $\Delta \xrightarrow{x} \Gamma$, then for every state $\Delta' \in \text{states}(\Gamma)$, put $\Delta \xrightarrow{x} \Delta'$.

⁴Note that, due to step 2 in the **(SR)** rule, every state contains at least one formula of the form $\exists \circ \varphi$.

4.4 State elimination phase

During this phase we remove from \mathcal{T}_0^θ states that are not satisfiable in a TEHS. As we do not create patently inconsistent states, there are two reasons why a state Δ of \mathcal{T}_0^θ can turn out to be unsatisfiable: either satisfiability of Δ requires satisfiability of some other (epistemic or temporal) successor states which turn out unsatisfiable, or Δ contains an eventuality that is not realized in the tableau. Accordingly, we have two elimination rules: **(E1)** and **(E2)**.

Formally, the state elimination phase is divided into stages; we start at stage 0 with \mathcal{T}_0^θ ; at stage $n + 1$, we remove exactly one state from the tableau \mathcal{T}_n^θ obtained at the previous stage, by applying one of the elimination rules, obtaining the tableau \mathcal{T}_{n+1}^θ . In the rules below, S_m^θ denotes the set of states of tableau \mathcal{T}_m^θ .

(E1) If Δ contains a formula χ of the form $\neg \mathbf{D}_A \varphi$ or $\exists \bigcirc \varphi$, and $\Delta \xrightarrow{\chi} \Delta'$ does not hold for any $\Delta' \in S_n^\theta$, obtain \mathcal{T}_{n+1}^θ by eliminating Δ from \mathcal{T}_n^θ .

For the other elimination rule, we need the concept of *eventuality realization* in a tableau.

Definition 4.1 (Eventuality realization)

- The eventuality $\neg \mathbf{C}_A \varphi$ is realized at Δ in \mathcal{T}_n^θ if there exists a path $\Delta = \Delta_0, \Delta_1, \dots, \Delta_m$, where $m \geq 0$, such that $\neg \varphi \in \Delta_m$ and, for every $0 \leq i < m$, there exist $\chi_i = \mathbf{D}_B \psi_i$ such that $B \subseteq A$ and $\Delta_i \xrightarrow{\chi_i} \Delta_{i+1}$.
- The eventuality $\exists(\varphi \mathcal{U} \psi)$ is realized at Δ in \mathcal{T}_n^θ if there exists a path $\Delta = \Delta_0, \Delta_1, \dots, \Delta_m$, where $m \geq 0$, such that $\psi \in \Delta_m$, and for every $0 \leq i < m$, there exist a formula χ_i such that $\Delta_i \xrightarrow{\exists \bigcirc \chi_i} \Delta_{i+1}$ and $\varphi \in \Delta_i$.
- For eventualities of the form $\forall(\varphi \mathcal{U} \psi)$, we define the notion “is realized at Δ in \mathcal{T}_n^θ ” recursively as follows:
 - (i) If $\psi \in \Delta$ then $\forall(\varphi \mathcal{U} \psi)$ is realized at Δ ;
 - (ii) If $\varphi \in \Delta$ and, for every $\exists \bigcirc \chi \in \Delta$, there is a state $\Delta' \in \mathcal{T}_n^\theta$ such that $\Delta \xrightarrow{\exists \bigcirc \chi} \Delta'$ and $\forall(\varphi \mathcal{U} \psi)$ is realized at Δ' , then $\forall(\varphi \mathcal{U} \psi)$ is realized at Δ .

Now, we can state our second state elimination rule.

(E2) If $\Delta \in S_n^\theta$ contains an eventuality ξ that is not realized at Δ in \mathcal{T}_n^θ , then obtain \mathcal{T}_{n+1}^θ by removing Δ from \mathcal{T}_n^θ .

We check for realization of eventualities by running the following iterative procedures that eventually marks all states realizing a given eventuality ξ in \mathcal{T}_n^θ :

- If $\xi = \neg \mathbf{C}_A \varphi$, then we initially mark all $\Delta \in S_n^\theta$ such that $\neg \varphi \in \Delta$. Then, we repeat the following sub-procedure until no more states get marked: for every

still unmarked $\Delta \in S_n^\theta$, mark Δ if there is at least one marked Δ' such that $\Delta \xrightarrow{\mathbf{D}_B \psi} \Delta'$, for some $B \subseteq A$.

- If $\xi = \exists(\varphi \mathcal{U} \psi)$, then we initially mark all $\Delta \in S_n^\theta$ such that $\psi \in \Delta$. Then, we repeat the following sub-procedure until no more states get marked: for every still unmarked $\Delta \in S_n^\theta$, mark Δ if $\varphi \in \Delta$ and there is at least one marked Δ' such that $\Delta \xrightarrow{\exists \bigcirc \xi} \Delta'$.
- If $\xi = \forall(\varphi \mathcal{U} \psi)$, then we initially mark all $\Delta \in S_n^\theta$ such that $\psi \in \Delta$, and then we repeat the following sub-procedure until no more states get marked: for every still unmarked $\Delta \in S_n^\theta$, mark Δ if $\varphi \in \Delta$ and, for every formula $\exists \bigcirc \chi \in \Delta$, there is a marked state $\Delta' \in S_n^\theta$ such that $\Delta \xrightarrow{\exists \bigcirc \chi} \Delta'$.

We now describe the order of application of the above rules. We have to be careful, since having applied **(E2)** to a tableau, we could have removed all the states accessible from some Δ along the arrows marked with some χ ; hence, we need to reapply **(E1)** to the resultant tableau to remove such Δ 's. Conversely, having applied **(E1)**, we could have thrown away states needed for realizing certain eventualities; hence, we need to reapply **(E2)**. Thus, we need to apply **(E1)** and **(E2)** in an alternating sequence that cycles through all eventualities. More precisely, we arrange all eventualities occurring in the states of \mathcal{T}_0^θ in a list ξ_1, \dots, ξ_m . Then, we proceed in cycles. Each cycle consists of alternately applying **(E2)** to the pending eventuality, starting with ξ_1 , and then applying **(E1)** to the resulting tableau, until all the eventualities have been dealt with, i.e., we have reached ξ_m . These cycles are repeated until no state is removed in a whole cycle. Then, the state elimination phase is over.

The graph produced at the end of the state elimination phase is called the *final tableau for θ* , denoted by \mathcal{T}^θ , whose set of states is denoted by S^θ .

Definition 4.2 The final tableau \mathcal{T}^θ is open if $\theta \in \Delta$ for some $\Delta \in S^\theta$; otherwise, \mathcal{T}^θ is closed.

If the final tableau is closed, the tableau procedure returns “no”; otherwise, it returns “yes”.

We briefly mention that, to test for satisfiability in general models, we relax the rule **(DR)**, allowing states to have epistemic successors from different temporal levels. As such a modification does not result in the outcome of the procedure, we conclude that, satisfiability-wise, the semantics based on general models is equivalent to the one based on synchronous models.

5 Soundness, completeness, and complexity

The *soundness* of a tableau procedure amounts to claiming that if the input formula θ is satisfiable, then the tableau

for θ is open. To establish soundness of the overall procedure, we use a series of lemmas showing that every rule by itself is sound; the soundness of the overall procedure is then an easy consequence. We give the proofs for the synchronous case, the modification for the general case being straightforward. The proofs of the following three lemmas are straightforward.

Lemma 5.1 *Let Γ be a pre-state of \mathcal{P}^θ such that $\mathcal{M}, (r, n) \Vdash \Gamma$ for some TEM \mathcal{M} and point (r, n) . Then, $\mathcal{M}, (r, n) \Vdash \Delta$ holds for at least one $\Delta \in \mathbf{states}(\Gamma)$.*

Lemma 5.2 *Let $\Delta \in S^\theta$ be such that $\mathcal{M}, (r, n) \Vdash \Delta$ for some TEM \mathcal{M} and point (r, n) , and let $\neg \mathbf{D}_A \varphi \in \Delta$. Then, there exists a point $(r', n) \in \mathcal{M}$ such that $((r, n), (r', n')) \in \mathcal{R}_A^D$ and $\mathcal{M}, (r', n') \Vdash \Delta'$ where $\Delta' = \{\neg \varphi\} \cup \bigcup_{A' \subseteq A} \{\mathbf{D}_{A'} \psi \mid \mathbf{D}_{A'} \psi \in \Delta\} \cup \bigcup_{A' \subseteq A} \{\neg \mathbf{D}_{A'} \psi \mid \neg \mathbf{D}_{A'} \psi \in \Delta\}$.*

Lemma 5.3 *Let $\Delta \in S^\theta$ be such that $\mathcal{M}, (r, n) \Vdash \Delta$ for some TEM \mathcal{M} and point (r, n) , and $\exists \bigcirc \varphi \in \Delta$. Then, $\mathcal{M}, (r', n+1) \Vdash \{\varphi\} \cup \{\psi \mid \forall \bigcirc \psi \in \Delta\}$ holds for some r' extending (r, n) .*

Lemma 5.4 *Let $\Delta \in S^\theta$ be such that $\mathcal{M}, (r, n) \Vdash \Delta$ for some TEM \mathcal{M} and a point (r, n) , and let $\neg \mathbf{C}_A \varphi \in \Delta$. Then, $\neg \mathbf{C}_A \varphi$ is realized at Δ in \mathcal{T}^θ .*

Proof idea. Since $\neg \mathbf{C}_A \varphi$ is true at s , there is a path in \mathcal{M} from s leading to a state satisfying $\neg \varphi$. As the tableau organizes an exhaustive search, a chain of tableau states corresponding to those states in the model will be produced. \square

The next two lemmas are proved likewise.

Lemma 5.5 *Let $\Delta \in S^\theta$ be such that $\mathcal{M}, (r, n) \Vdash \Delta$ for some TEM \mathcal{M} and a point (r, n) , and let $\exists(\varphi \mathcal{U} \psi) \in \Delta$. Then, $\exists(\varphi \mathcal{U} \psi)$ is realized at Δ in \mathcal{T}^θ .*

Lemma 5.6 *Let $\Delta \in S^\theta$ be such that $\mathcal{M}, (r, n) \Vdash \Delta$ for some TEM \mathcal{M} and a point (r, n) , and let $\forall(\varphi \mathcal{U} \psi) \in \Delta$. Then, $\forall(\varphi \mathcal{U} \psi)$ is realized at Δ in \mathcal{T}^θ .*

Theorem 5.7 *If $\theta \in \mathcal{L}$ is satisfiable in a TEM, then \mathcal{T}^θ is open.*

Proof sketch. Using the preceding lemmas, we show by induction on the number of stages in the state elimination phase that no satisfiable state can be eliminated due to (E1) or (E2). The claim then follows from Lemma 5.1. \square

The *completeness* of a tableau procedure means that if the tableau for a formula θ is open, then θ is satisfiable in a TEM. In view of Theorem 3.7, it suffices to show that an open tableau for θ can produce a TEHS satisfying θ . Moreover, we show that this TEHS can be constructed synchronous.

Lemma 5.8 *If \mathcal{T}^θ is open, then θ is satisfiable in a (synchronous) TEHS.*

Proof sketch. We build the TEHS \mathcal{H} for θ by induction on the temporal levels. The main concern is to ensure that all eventualities in the resultant structure are realized, i.e. the properties (H5), (H6) and (H9) from the definition of Hintikka structures hold; all the other properties of Hintikka structures transfer, more or less immediately, from an open tableau. We alternate between realizing epistemic eventualities (formulae of the form $\neg \mathbf{C}_A \varphi$) and temporal eventualities (formulae of the form $\exists(\varphi \mathcal{U} \psi)$ and $\forall(\varphi \mathcal{U} \psi)$). Essentially, the construction combines the construction used in proving completeness of multi-agent epistemic Hintikka structures from [6] and the one used in proving completeness of CTL (see [10], which essentially uses the construction that is a simplification of the construction for ATL from [4]).

We start by building the 0th level of our prospective Hintikka structure from the level 0 of an open tableau. For each state $\Delta^{[0]}$ on this level, if $\Delta^{[0]}$ does not contain any epistemic eventualities, we define $\Delta^{[0]}$ -epistemic component to be $\Delta^{[0]}$ with exactly one successor reachable by $\neg \mathbf{D}_A \psi$, for each $\neg \mathbf{D}_A \psi \in \Delta^{[0]}$; if, on the other hand, $\neg \mathbf{C}_A \varphi \in \Delta^{[0]}$, then the $\Delta^{[0]}$ -epistemic component is a tree obtained from a path in the tableau leading from $\Delta^{[0]}$ along the arrows marked with formulae of the form $\neg \mathbf{D}_B \chi$ to a state $\Delta'^{[0]}$ containing φ ; the tree is obtained from the path by giving each component of the path enough successors, as described above. As all the unrealized epistemic eventualities are propagated down the components (hence, appear in the leaves of the tree), we can stitch them up together to obtain a graph where each epistemic eventuality is realized.

Now, having built the 0th level of our prospective Hintikka structure, we take care of realizing all the temporal eventualities contained in the states of level 0. This is done exactly as in the completeness proof of the tableau procedure for CTL ([10]): we define the $\Delta^{[0]}$ -temporal component for each $\Delta^{[0]}$ as follows: if it does not contain any temporal eventualities, then we take $\Delta^{[0]}$ with one temporal successor for each $\exists \bigcirc \varphi \in \Delta$. If $\exists(\varphi \mathcal{U} \psi) \in \Delta$, then we take a temporal path realizing $\exists(\varphi \mathcal{U} \psi) \in \Delta$ and give to every node enough temporal successors, as describe above. Lastly, if $\forall(\varphi \mathcal{U} \psi) \in \Delta$, then we take a temporal tree witnessing the the realization of $\forall \varphi \in \Delta$ in the tableau (for details, see [10]). As eventualities are again passed down, we can stitch up an infinite tree realizing all the eventualities contained in the states making up the tree.

Next, we repeat the procedure inductively. For the m th epistemic level, we independently apply to each state on this level the procedure described above for level 0, so that the epistemic structures unfolding from any two points on level m are disjoint, and also give to each newly created point a 'history' consisting of a path of $m-1$ states of the form $\{\top\}$

(so that we do not create any new epistemic eventualities). Having fixed all the epistemic eventualities at the m th level, we repeat the procedure described in the previous paragraph to fix all the temporal eventualities contained in states of level m .

Thus, we produce a chain of structures ordered by inclusion. Eventually, we take the (infinite) union of all the structures defined at the finite states of that construction, and then put $H(\Delta^{[k]}) = \Delta^{[k]}$ for every $\Delta^{[k]}$, to obtain a TEHS satisfying θ . \square

The completeness is now immediate from Lemma 5.8 and Theorem 3.7.

Theorem 5.9 (Completeness) *If T^θ is open, then θ is satisfiable, in a (synchronous) TEM.*

As for the complexity, for lack of space, we only mention that the above procedure runs in exponential time (the calculations are fairly routine), thus matching the lower bound known from [7].

6 Concluding remarks

We have presented an incremental tableau-based decision procedure for the full coalitional temporal-epistemic logic of branching time **CMATEL(CD+BT)**. The procedure is complexity-optimal, intuitive, and practically reasonably efficient (as the number of (pre)states it creates is usually significantly smaller than the powerset of all subsets of the close of the formula that is tested for satisfiability); it is, therefore, suitable for both manual and automated execution. Moreover, it is fairly flexible and easily amenable to modifications for variations of the semantics, such as those mentioned in section 2. Since in the semantics considered in this paper there is essentially no interaction between the temporal and epistemic fragments, our procedure combines in a modular way tableaux for the full coalitional multiagent epistemic logic **CMAEL(CD)** and for **CTL**. Such interaction, however, can be triggered by imposing various natural semantic conditions, such as “no learning” or “no forgetting”. As shown in [7], such conditions may increase dramatically the complexity of the logic, up to highly undecidable. However, even for the relatively ‘easy’ cases of EXPSPACE-hard logics, the construction of a tableau procedure is still an open challenge, which we intend to address in the future.

References

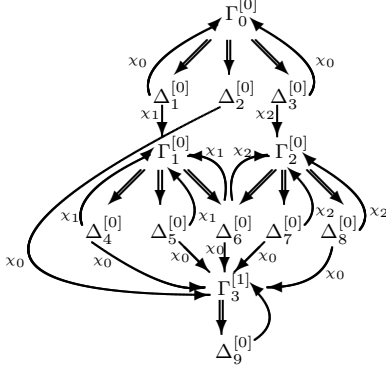
[1] Mordechai Ben-Ari, Amir Pnueli, and Zohar Manna. The temporal logic of branching time. *Acta Informatica*, 20:207–226, 1983.

- [2] E. Allen Emerson. Temporal and modal logics. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. MIT Press, 1990.
- [3] E. Allen Emerson and Joseph Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *Journal of Computer and System Sciences*, 30(1):1–24, 1985.
- [4] Valentin Goranko and Dmitry Shkatov. Tableau-based decision procedures for logics of strategic ability in multi-agent systems. To appear in *ACM Transactions on Computational Logic*. Available at <http://tocl.acm.org/accepted.html>.
- [5] Valentin Goranko and Dmitry Shkatov. Tableau-based decision procedure for full coalitional multiagent temporal-epistemic logic of linear time. In Decker, Sichman, Sierra, and Castelfranchi, editors, *Proc. of the 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009), May 2009, Budapest, Hungary, 2009*.
- [6] Valentin Goranko and Dmitry Shkatov. Tableau-based procedure for deciding satisfiability in the full coalitional multiagent epistemic logic. In Sergei Artemov and Anil Nerode, editors, *Proc. of the Symposium on Logical Foundations of Computer Science (LFCS 2009)*, volume 5407 of *Lecture Notes in Computer Science*, pages 197–213. Springer-Verlag, 2009.
- [7] Joseph Y. Halpern and Moshe Y. Vardi. The complexity of reasoning about knowledge and time I: Lower bounds. *Journal of Computer and System Sciences*, 38(1):195–237, 1989.
- [8] Agi Kurucz, Frank Wolter, Michael Zakharyashev, and Dov Gabbay. *Multi-dimensional Modal Logics: Theory and Applications*. Elsevier, 2003.
- [9] Vaughan R. Pratt. A practical decision method for propositional dynamic logic. In *Proceedings of the 10th Annual ACM Symposium on the Theory of Computing*, pages 326–227, San Diego, California, May 1979.
- [10] Dmitry Shkatov. Incremental **CTL**-tableaux revisited. Submitted.
- [11] Dirk Walther. ATEL with common and distributed knowledge is ExpTime-complete. In *Proceedings of Methods for Modalities 4*, Berlin, 2005.
- [12] Dirk Walther, Carsten Lutz, Frank Wolter, and Michael Wooldridge. ATL satisfiability is indeed ExpTime-complete. *Journal of Logic and Computation*, 16(6):765–787, 2006.

- [13] Pierre Wolper. The tableau method for temporal logic: an overview. *Logique et Analyse*, 28(110–111):119–136, 1985.

A Example

In the present appendix, we provide an example of how our procedure works on the formula $\forall(\neg C_{\{a,b\}}p \mathcal{U} \neg D_{\{a,c\}}p)$. To simplify the example, we test for satisfiability of the equivalent set $\{\forall(\neg C_{\{a,b\}}p \mathcal{U} \neg D_{\{a,c\}}p, \top)\}$. Displayed below is the complete pretableau for this set.



$\chi_0 = \exists \top$;
 $\chi_1 = \neg D_a(p \wedge C_{\{a,b\}}p)$;
 $\chi_2 = \neg D_b(p \wedge C_{\{a,b\}}p)$;
 $\Gamma_0 = \{\forall(\neg C_{\{a,b\}}p \mathcal{U} \neg D_{\{a,c\}}p) = \theta, \top\}$;
 $\Delta_1 = \{\theta, \neg C_{\{a,b\}}p, \forall \theta, \chi_1, \exists \top\}$;
 $\Delta_2 = \{D_{\{a,c\}}p, p, \exists \top\}$;
 $\Delta_3 = \{\theta, \neg C_{\{a,b\}}p, \forall \theta, \chi_2, \exists \top\}$;
 $\Gamma_1 = \{\chi_1, \neg(p \wedge C_{\{a,c\}}p)\}$;
 $\Gamma_2 = \{\chi_2, \neg(p \wedge C_{\{a,c\}}p)\}$;
 $\Delta_4 = \{\chi_1, \neg p, \exists \top\}$;
 $\Delta_5 = \{\chi_1, \neg C_{\{a,b\}}p, \exists \top\}$;
 $\Delta_6 = \{\chi_1, \neg C_{\{a,b\}}p, \chi_2, \exists \top\}$;
 $\Delta_7 = \{\chi_2, \neg C_{\{a,c\}}p, \exists \top\}$;
 $\Delta_8 = \{\chi_2, \neg p, \exists \top\}$;
 $\Gamma_3 = \{\top\}$;
 $\Delta_9 = \{\top, \exists \top\}$.

The initial tableau is obtained by removing all pre-states (the Γ s) and redirecting the arrows (i.e. Δ_1 will be connected by unmarked single arrows to itself, Δ_2 , and Δ_3). It is easy to check that no states get removed during the state elimination stage; hence, the tableau is open and θ is satisfiable.

A framework to model norm dynamics in Answer Set Programming

Sofia Panagiotidi and Juan Carlos Nieves and Javier Vázquez-Salceda
 Knowledge Engineering and Machine Learning Group
 Universitat Politècnica de Catalunya, Spain
 {panagiotidi,jcnieves,jvazquez}@lsi.upc.edu

Abstract—This paper presents ongoing work in the formal definition and implementation of a normative framework in Answer Set Programming. The framework uses as basis an existing action language and enriches it with a formal definition of norms (implementing standard deontic operators such as obligations and permissions). Properties of a norm's lifecycle such as active, inactive, violated are specified. We argue that such properties can serve as a reasoning basis for the agent's cycle. A partial implementation of the framework is then presented. An example is used in order to illustrate the application of the principles presented.

I. INTRODUCTION

The field of normative systems is an active area where researchers try to find formalisations and mechanisms to model and reason about normative statements. There is a lot of formal work on the formalisation of norms, typically based in variants of monadic or dyadic Deontic Logic. In this way, normative systems predefine the desired behaviour in terms of deontic concepts (obligations, prohibitions, permissions), which typically are extended with other concepts such as deadlines, violations and sanctions. But there is few work on how to bring these theories into practice.

One of the main barriers for the use of deontic-like formalisations is the lack of operational semantics [1]. Without clear operational semantics an agent cannot clearly reason about the influence of these norms in its practical reasoning mechanism. Some attempts have been made to reduce deontic formalisations of norms into operational formalisations based on, e.g. dynamic Logic [2] or Linear Temporal Logic [3], but often such formalisations cannot really be used at execution time.

Other attempts focus on extending existing operational formalisms with some normative concepts. These attempts mainly concern the design of action languages based on the effects of axioms and drawing inferences from the axioms and concentrate on ways of reasoning about actions. [4] presents an action description language capable of expressing causal laws which describe effects of actions as well as statements about values of fluents in possible states of the world. Work of the same authors covers some aspects of reasoning over dynamic domains [5] and representation and reasoning over properties of actions [6]. In [7] the authors intent to provide a logic based language in order to represent authorizations and obligation policies within dynamic environments as well as methods for checking compliance of performed actions with the defined

policies. Still, the work lacks support for time. Insight into intelligent decision and possible prediction of desired or non-desired behaviours in advance seems to be absent too.

We are aware of the work in [8] in which the authors present a language for implementing multi-agent systems consisting of individual agents that interact with a computational organization specified in terms of roles, norms and sanctions. Still, this piece of work adopts an organisation-centered view more on the flexible role enactment within a BDI cycle while ours assumes a static role enactment and focuses on a decision making mechanism inferring possible future action taking and violated states.

In the last two decades, one of the most successful logic programming approach has been Answer Set Programming (ASP) [9]. ASP is the realization of much theoretical work on Non-monotonic Reasoning and Artificial Intelligence applications. It represents a new paradigm for logic programming that allows, using the concept of *negation as failure*, to handle problems with default knowledge and produce non-monotonic reasoning. Since its inception, ASP has been regarded as the computational embodiment of Nonmonotonic Reasoning and a primary candidate for an effective knowledge representation tool. This view has been boosted by the emergence of highly efficient solvers for ASP [10], [11], [12]. The efficiency of the answer set solvers has allowed to increase the list of ASP's practical applications, e.g., planning [13], Bioinformatics [14], [15], argumentation theory [16], etc. Answer Set Programming does not represent a conventional logic programming approach as PROLOG. Usually an answer set program P can be regarded as a specification of a given problem where each answer set (a model) of P represents a possible solutions of the given problem.

In this paper (Figure 1) we propose, based on a previous normative formalisation [17], a methodology to represent such normative frameworks in Answer Set Programming. This representation can be used as a basis for decision making procedures as well as planning methodologies. We use as example the obligations between two actors (a Insurance Company and a Repair Company) defined in a electronic contract. This contract is composed by deontic clauses which will become norms of the expected behaviour of the contractual parties. Therefore each instantiated contract becomes a normative context at execution time.

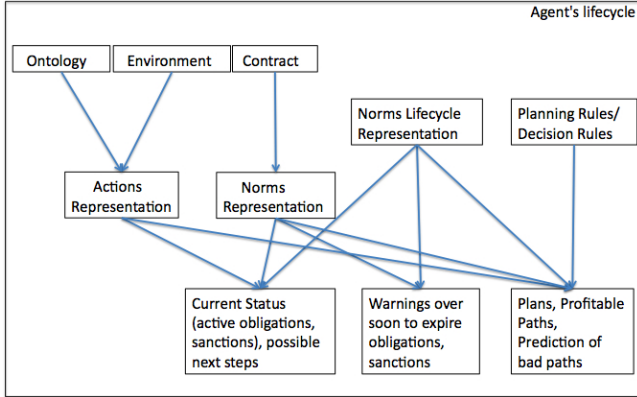


Fig. 1. Architecture of the normative framework implementation

The work is at a preliminary state. Still, integrated within an agent's operating cycle, such a framework can operate as the brain of the agent [18], providing the necessary information over the current state of the world such as pending norms and currently violated obligations. Additional layers of planning rules and preferred choices might lead to the desired intelligence and flexibility an agent needs to perform in a normative environment. We define several layers of abstraction (Figure 1). An agent (possibly interacting with an ontology) can interpret the environment it operates in through an action language. The deontic restrictions (which in this paper come from a contract, but in other scenario could come from a Institutional specification) provide guidelines over the agent's behaviour and are formalised through a deontic framework.

Generic rules over the interpretation of norms, the interpretation of the action language and planning rules defined with respect to the action language might be added on top of the previous in order to be able to extract information over the agent's state, know the possible actions to take, predict possible violations and foresee future paths. Our interest and focus can be summarised in the following four:

- An action language representation
- Contract (viewed as a set of norms) representation
- Generic rules over the norm lifecycle
- Generic planning and choice rules

Currently, the three first of the above are implemented while the fourth and equally interesting issue is under research. In this paper we are aiming to provide the functionality of the aforementioned as well as some preliminary planning rules. We argue that the implemented layers leave further space for future research over the agent's planning capabilities and behavioural choices.

The rest of the paper is structured as follows. Section II defines the stable model semantics which is the basis for Answer Set Programming. In section III we present a simple motivating example which supports our work while in section

IV we detail the action and normative language used and define formal properties of the framework with respect to the aforementioned. A partial implementation of the action and normative language in Answer Set Programming is provided in sections V and VI and parts of the motivating example presented in the beginning are cited in section VIII. We conclude in section IX and summarise future work.

II. BACKGROUND

In this section, we present some basic definitions *w.r.t.* normal programs and the stable model semantics.

A. Syntax

A signature \mathcal{L} is a finite set of elements that we call atoms. A *literal* is either an atom a , called *positive literal*; or the negation of an atom $\text{not } a$, called *negative literal*. Given a set of atoms $\{a_1, \dots, a_n\}$, we write $\text{not } \{a_1, \dots, a_n\}$ to denote the set of atoms $\{\text{not } a_1, \dots, \text{not } a_n\}$. A *normal clause*, C , is a clause of the form

$$a \leftarrow b_1, \dots, b_n, \text{not } b_{n+1}, \dots, \text{not } b_{n+m}$$

where a and each of the b_i are atoms for $1 \leq i \leq n + m$. In a slight abuse of notation we will denote such a clause by the formula $a \leftarrow \mathcal{B}^+ \cup \text{not } \mathcal{B}^-$ where the set $\{b_1, \dots, b_n\}$ will be denoted by \mathcal{B}^+ , and the set $\{b_{n+1}, \dots, b_{n+m}\}$ will be denoted by \mathcal{B}^- . We define a *normal program* P , as a finite set of normal clauses.

If the body of a normal clause is empty, then the clause is known as a *fact* and can be denoted just by $a \leftarrow \cdot$ or simply a . We write \mathcal{L}_P , to denote the set of atoms that appear in the clauses of P . We denote by $HEAD(P)$ the set $\{a | a \leftarrow \mathcal{B}^+, \text{not } \mathcal{B}^- \in P\}$.

B. Stable model semantics.

The stable model semantics was defined in terms of the so called *Gelfond-Lifschitz reduction* [19] and it is usually studied in the context of syntax dependent transformations on programs. The following definition of a stable model for normal programs was presented in [19]:

Let P be any normal program. For any set $S \subseteq \mathcal{L}_P$, let P^S be the definite program obtained from P by deleting

- (i) each rule that has a formula $\text{not } l$ in its body with $l \in S$, and then
- (ii) all formulæ of the form $\text{not } l$ in the bodies of the remaining rules.

Clearly P^S does not contain not . Hence S is a stable model of P if and only if S is a minimal model of P^S .

In order to illustrate this definition let us consider the following example:

Let $S = \{b\}$ and P be the following logic program:

$$\begin{array}{ll} b \leftarrow \text{not } a. & b \leftarrow \top. \\ c \leftarrow \text{not } b. & c \leftarrow a. \end{array}$$

We can see that P^S is:

$$b \leftarrow \top. \qquad c \leftarrow a.$$

Notice that P^S has three models: $\{b\}$, $\{b, c\}$ and $\{a, b, c\}$. Since the minimal model amongst these models is $\{b\}$, we can say that S is a stable model of P .

III. EXAMPLE

The focus of interest of this example is to show how a contract and a set of instantiated norms over the repair of a car operate within the domain and normative environment implemented in ASP. Such a case is useful to demonstrate how reasoning and planning over the norms defined in a contract can be done throughout its execution.

Our scenario is as follows: Two actors, a Repair Company and a Client create a contract (*RepairContract*) over the repair of a broken car. Initially, the Client takes the car to the Repair Company. Once the car is there, the Repair Company has to repair it. Whenever it is repaired, the Client has to pay a previously agreed price. Once the car is repaired, the client has the right to make a complaint.

Figure 2 depicts a pseudo-diagram including the interesting states that the process will go into.

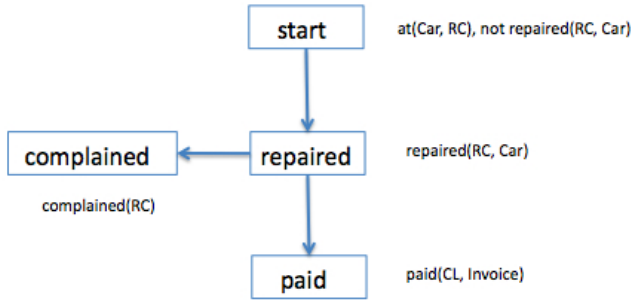


Fig. 2. Pseudo states diagram

The focus of interest of this example is to show how the norms of an electronic contract (explicitly expressed as clauses in the contract) can be mapped into ASP rules and, in a combination with an appropriate action language, to reason over, plan future actions and detect violations within the agent's scope.

IV. LANGUAGE USED

A. Actions

The action description in our approach is done as in most action description languages. In this work we use the formalisation of [6] but adapt several rules to fit our requirements.

Thus the alphabet of the action language \mathcal{A} consists of two nonempty disjoint sets of symbols F and A . They are called the set of fluents and the set of actions. Fluents express the property of an object in a state of the world. A fluent or *fluent literal* (e.g. *car_at_shop*) is a *positive literal* or a *strongly negated literal* (a fluent preceded by \neg). A *state* σ is a collection of fluents. We say a fluent f holds in a state σ if $f \in \sigma$. We say a fluent literal $\neg f$ holds in σ if $f \notin \sigma$.

An action domain description D within the action language \mathcal{A} consists of effect propositions of the following form:

$$a \text{ causes } f \text{ if } p_1, \dots, p_n, \neg q_1, \dots, \neg q_r \quad (4.1)$$

where a is an action in A , f is a fluent literal in F (also

called *postcondition* of the action a) and $p_1, \dots, p_n, q_1, \dots, q_r$ are fluents (also called *preconditions* of the action a) in F .

We assume a set of initial observations O that consists of value propositions of the following form:

$$f \text{ after } a_1, \dots, a_m \quad (4.2)$$

where f is a fluent literal and a_1, \dots, a_m are actions. Intuitively that means that f will hold after the execution of a_1, \dots, a_m .

When a_1, \dots, a_m is an empty sequence then we write:

$$\text{initially } f \quad (4.3)$$

We also condense a set of value propositions of the form $\{\text{initially } f_1, \dots, \text{initially } f_n\}$ by

$$\text{initially } f_1, \dots, f_n \quad (4.4)$$

The role of effect propositions is to define a transition function from states and actions to states. Given a domain description D , such a transition function Φ should satisfy the following properties. For all actions α , fluents g , and states σ :

- if D includes an effect proposition of the form (4.1) where f is the fluent g and $p_1, \dots, p_n, \neg q_1, \dots, \neg q_r$ hold in σ then $g \in \Phi(\alpha, \sigma)$;
- if D includes an effect proposition of the form (4.1) where f is a negative fluent literal $\neg g$ and $p_1, \dots, p_n, \neg q_1, \dots, \neg q_r$ hold in σ then $g \notin \Phi(\alpha, \sigma)$;
- if D does not include such effect propositions then $g \in \Phi(\alpha, \sigma)$ iff $g \in \sigma$;

If such a transition function exists, then we say that D is consistent, and refer to its transition function by Φ_D . Given a consistent domain description D the set of observations O is used to determine the states corresponding to the initial situation, referred to as *the initial states* and denoted by σ_0 . While D determines a unique transition function, an O may not always lead to a unique initial state.

We say σ_0 is an initial state corresponding to a consistent domain description D and a set of observations O , if for all observations of the form (4.2) in O , the fluent literal f holds in the state $\Phi(\alpha_m, \Phi(\alpha_{m-1}, \dots, \Phi(\alpha_1, \sigma_0) \dots))$ (We will denote this state by $[\alpha_m, \dots, \alpha_1]\sigma_0$). We then say that (σ_0, Φ_D) *satisfies* O .

Given a consistent domain description D and a set of observations O , we refer to the pair (Φ_D, σ_0) where Φ_D is the transition function of D and σ_0 is an initial state corresponding to (D, O) as a *model* of (D, O) . We say (D, O) is *consistent* if it has a model and say it is *complete* if it has a unique model.

We say a consistent domain description D in the presence of a set of observations O entails a query Q of the form (4.2) if for all initial states σ corresponding to (D, O) , the fluent literal f holds in the state $[\alpha_m, \dots, \alpha_1]\sigma_0$. We denote this as $D \models_O Q$.

Example 1: With respect to our example, let's assume that the set of actions $A = \{\text{repair}(\text{Actor}), \text{pay}(\text{Actor}), \text{complain}(\text{Actor})\}$, the set of fluents

$$F = \{\text{car_at_shop}, \text{repaired}(\text{Actor}), \text{paid}(\text{Actor}), \text{complained}(\text{Actor})\}$$

and that the domain description consists of the following:

$\text{repair}(\text{Actor})$ **causes** $\text{repaired}(\text{Actor})$ **if** car_at_shop

$\text{pay}(\text{Actor})$ **causes** $\text{paid}(\text{Actor})$

$\text{complain}(\text{Actor})$ **causes** $\text{complained}(\text{Actor})$

Also suppose we have a set of observations

$$O = \{\text{initially } \text{car_at_shop}, \neg \text{repaired}(rc), \neg \text{paid}(cl), \neg \text{complained}(cl)\}. \square$$

A set of observations O is said to be initial state complete, if O only consists of propositions of the form (4.3) and for all fluents, either **initially** f is in O , or **initially** $\neg f$ is in O , but not both.

B. Deontic statements

Norms (clauses) express agreements between parties in the form of deontic statements. In order to express the clauses we have adopted a variation [20] [17] of the norm representation defined in [3].

Since norms may have normative force only in certain situations, they are associated with an activation condition. Norms are thus typically abstract, and are instantiated when the norm's activation condition holds. Once a norm has been instantiated, it remains active, irrespective of its activation condition, until a specific expiration condition holds. When the expiration condition occurs, the norm is assumed no longer to have normative force. Independent of these two conditions is the norm's normative goal, which is used to identify when the norm is violated (in the case of an obligation), or what the agent is actually allowed to do (in the case of a permission). Obligations and permissions are the two norm types on which our framework focuses. For every norm there is a maintenance condition which, if does not hold while the norm is active then a violation is considered to be taking place. Finally, for every norm there exists another one (violation norm) indicating what happens

if it gets violated. Such properties of norms with respect to a normative environment are implemented in section VI.

We assume a set of norms DS . A norm N in DS is represented as a tuple of the form [17]:

$$\langle \text{NormType}, \text{DeonticStatement}, \text{ActivatingCondition}, \text{MaintenanceCondition}, \text{DisactCondition}, \text{Actor} \rangle \text{ (4.5)}$$

such that:

- $\text{NormType} \in \{\text{obligation}, \text{permission}\}$
- DeonticStatement is an action a in the set of actions A_{norm} subset of A
- an $\text{ActivatingCondition}$ is of the type: $k_1, \dots, k_n, \neg l_1, \dots, \neg l_n$, where $k_1, \dots, k_n, l_1, \dots, l_n$ are fluents in F
- a DisactCondition is of the type: $u_1, \dots, u_n, \neg v_1, \dots, \neg v_n$, where $u_1, \dots, u_n, v_1, \dots, v_n$ are fluents in F
- a $\text{MaintenanceCondition}$ is of the type: $w_1, \dots, w_n, \neg z_1, \dots, \neg z_n$, where $w_1, \dots, w_n, z_1, \dots, z_n$ are fluents in F
- a Actor is the actor responsible for the deontic statement

The definition and semantics of permissions are not trivial. For example, it is dubious whether their existence can be assumed within the context of an environment without obligations. In [21] the authors discuss the different types of permissions and their role in deontic logic. We assume that permissions are an explicit indication on what an actor can perform. As in standard deontic logic we assume further that whenever there is an obligation it is always implied that the same is a permission too ($O_a \rightarrow P_a$, axiom D of deontic logic).

Example 2: Figure 3 depicts the formal representation of the deontic statements within the contract of the example presented in the third section. \square

Below we give some definitions with respect to the action language and the deontic statements. Let a norm N be in the form of (4.5) and σ be a state.

- N is activated in σ iff $k_1, \dots, k_n, \neg l_1, \dots, \neg l_n$ hold in σ . We denote this as $D \models_{DS, O, \sigma} \text{activated}(N)$. In the contrary case, N is not activated. We denote this as $D \models_{DS, O, \sigma} \neg \text{activated}(N)$.
- N is disactivated in σ iff $u_1, \dots, u_n, \neg v_1, \dots, \neg v_n$ hold in σ . We denote this as $D \models_{DS, O, \sigma} \text{disactivated}(N)$. In the contrary case, N is not disactivated. We denote this as $D \models_{DS, O, \sigma} \neg \text{disactivated}(N)$.
- N is violated in σ iff at least one of $w_1, \dots, w_n, \neg z_1, \dots, \neg z_n$ does not hold in σ . We denote this as $D \models_{DS, O, \sigma} \text{violated}(N)$. In the contrary case, N is not violated. We denote this as $D \models_{DS, O, \sigma} \neg \text{violated}(N)$.

Clause	<i>RC1: Repair Company has to repair the car within four days after the car arrives.</i>
Actor	Repair Company
Modality	obligation
Activating Condition	car_at_shop, -repaired(rc)
Maintenance Condition	before(4days)
Deontic Statement	repair(rc)
Disactivating Condition	repaired(rc)

Clause	<i>RC2: The Client has the obligation to pay the pre-agreed price two days within the end of the repair of the car.</i>
Actor	Client
Modality	obligation
Activating Condition	repaired(rc), -paid(rc)
Maintenance Condition	before(2days)
Deontic Statement	pay(cl)
Disactivating Condition	paid(cl)

Clause	<i>RC3: The Client has the right to complain seven days within the end of the repair of the car.</i>
Actor	Client
Modality	permission
Activating Condition	repaired(rc)
Maintenance Condition	before(7days)
Deontic Statement	complain(cl)
Disactivating Condition	complained(cl)

Fig. 3. Formal representation of norms in the Repair Contract

Queries with respect to norms consist of value propositions of the form:

$$\begin{aligned}
 & \text{activated}(N) \text{ after } a_1, \dots, a_m \\
 & \text{disactivated}(N) \text{ after } a_1, \dots, a_m \\
 & \text{violated}(N) \text{ after } a_1, \dots, a_m
 \end{aligned} \quad (4.6)$$

where N is a deontic statement of the form (4.5), and a_1, \dots, a_m are actions in A .

We say a consistent domain description D in the presence of a set of observations O and a set of norms DS entails a query Q of the form (4.6) if for all initial states σ_0 corresponding to (D, O) the proposition $\text{activated}(N)$, $\text{disactivated}(N)$, $\text{violated}(N)$ holds respectively in the state $[\alpha_m, \dots, \alpha_1]\sigma_0$. We denote this as $D \models_{DS, O} Q$.

Example 3: Let's assume the set of actions A , the set of fluents F and the set of observations O in example 1

and the set of norms DS in example 2. O is an initial state complete. Then the initial state corresponding to (D, O) will be

$$\sigma_0 = \{\text{initially car_at_shop}, \neg \text{repaired}(rc), \neg \text{paid}(cl), \neg \text{complained}(cl)\}.$$

Then:

$D \models_{DS, O} \text{activated}(N_{RC2})$ since $\text{activated}(N_{RC2})$ holds in the state $[\text{repair}(rc)]\sigma_0$.

$D \models_{DS, O} \text{activated}(N_{RC3})$ since $\text{activated}(N_{RC3})$ holds in the state $[\text{repair}(rc)]\sigma_0$.

where N_{RC2} is the norm $RC2$ and N_{RC3} is the norm $RC3$. \square

Proposition 1: Let D be a consistent domain description, O be an initial state complete set of observations and DS a set of deontic statements. If (D, O) is consistent then $\forall N \in DS$ and for any state σ reached by Φ_D :

- $D \models_{DS, O, \sigma} \text{activated}(N)$ or $D \models_{DS, O, \sigma} \neg \text{activated}(N)$
- $D \models_{DS, O, \sigma} \text{disactivated}(N)$ or $D \models_{DS, O, \sigma} \neg \text{disactivated}(N)$
- $D \models_{DS, O, \sigma} \text{violated}(N)$ or $D \models_{DS, O, \sigma} \neg \text{violated}(N)$

C. Temporal Projection

Planning: In the case of planning we are given a domain description D , a set of observations about the initial state O , a collection of deontic statements DS and a collection of fluent literals $G = \{g_1, \dots, g_h\}$ which we will refer to as goal. We are required to find a sequence of actions a_1, \dots, a_n such that for all $1 \leq i \leq h$, $D \models_O g_i$ **after** a_1, \dots, a_n and for all states σ passed throughout the execution of the actions, $\nexists \sigma$ such that $D \models_{DS, O, \sigma} \text{violated}(N)$.

V. REPRESENTATION OF ACTIONS

Given an action domain description D , a set of observations O and a set of deontic statements DS we construct a program that partially implements the above. The basic elements and definitions of types are (π_{basic}) . The program putting into effect (D, O) consists of three parts namely π_{ef} , π_{obs} , π_{in} which represent the effect propositions, observations and inertia rules. In addition, π_{norm} and (π_{lc}) partly implement the norms specification and norm lifecycle. We denote the union of $\pi_{basic} \cup \pi_{ef} \cup \pi_{obs} \cup \pi_{in} \cup \pi_{norm} \cup \pi_{lc}$ as π .

In addition to the approach of [6], we assume that actions are durative. That is, they have a starting and ending time and a duration which is the interval between the two. For this, we use predicates like *started* and *terminated* to indicate the time that an action starts being executed and the time it

terminates.

A. Basic elements

We introduce here some predicates (π_{basic}) which avoid the program to have weakly restricted rules.

(1) The predicate *fluent*. For every fluent $f \in F$ we add the following rule to the program.

$$fluent(f) \leftarrow .$$

(2) The predicate *action*. For every action $a \in A$ we add a rule:

$$action(a) \leftarrow .$$

(3) The predicate *time*. The predicate is used to represent number of distinct times which in their turn represent the time flow. Initially $time(T)$ where T is a constant.

B. Action rules

(1) Translating effect propositions (π_{ef}). The effect propositions in D are translated as follows. For every effect proposition of the form (4.1) if f is a fluent in F then the following rules are created:

$$holds(f, T + 1) \leftarrow terminated(a, T), time(T).$$

$$ab(f, a, T) \leftarrow terminated(a, T), time(T).$$

else if f is the negative fluent literal $\neg g$ then the following rules are created:

$$\neg holds(g, T + 1) \leftarrow terminated(a, T), time(T).$$

$$ab(g, a, T) \leftarrow terminated(a, T), time(T).$$

Intuitively, predicate *holds* indicates whether a fluent's value is true or false throughout time. The *ab* predicate represents changes of a fluent's value at some specific timestep.

(2) Translating observations (π_{obs}). We intentionally omit observations of the type (4.2) and assume that all observations are of the type (4.3). The value propositions in O then are translated as follows. For every value proposition of the form (4.3) if f is a positive fluent literal then the following rule is made:

$$holds(f, 1) \leftarrow .$$

else if f is the negative fluent literal $\neg g$ then the following rule is created:

$$\neg holds(g, 1) \leftarrow .$$

(3) Inertia rules (π_{in}). Besides the above, we have the following inertia rules. The first one makes sure that if no action is executed during a timestep then a fluent will keep holding a true value at the next timestep if it already holds true at the current timestep. The second one makes sure that if a fluent holds a true value and the action that is performed at some timestep does not make it false, then in the next timestep it will also hold true. The third one says that if no action is executed during a timestep then a fluent will keep holding a false value at the next timestep if it already holds false at the current timestep. The fourth one makes sure that if a fluent holds a false value and the action that is performed at some timestep does not make it true, then in the next timestep it will also hold false.

$$holds(F, T + 1) \leftarrow holds(F, T), not\ terminated(A, T).$$

$$holds(F, T + 1) \leftarrow holds(F, T), terminated(A, T), not\ ab(F, A, T).$$

$$\neg holds(F, T + 1) \leftarrow not\ holds(F, T), not\ terminated(A, T).$$

$$\neg holds(F, T + 1) \leftarrow not\ holds(F, T), terminated(A, T), not\ ab(F, A, T).$$

The following proposition demonstrates how the *holds* predicate of π will hold a true value at some time step if f holds true at some state σ reached by the function Φ_D .

Proposition 2: Let D be a consistent domain description and O be an initial complete set of observations such that (D, O) is consistent. Let (σ_0, Φ_D) be the unique model of (D, O) and M a stable model of π . If f is a fluent in F then

$f \in \Phi_D(\alpha_n, \dots, \Phi_D(\alpha_1, \sigma_0))$ iff $holds(f, T) \in M$ for some $T \in \mathbb{N}$.

(4) Executability rules. It is assumed that every action might be executed whenever the conditions are fulfilled. We use the predicate *executable* to model the state where the preconditions of action are fulfilled and the action might be executed.

Thus, for every effect proposition of the form (4.1) the following is created:

$$executable(a, T) \leftarrow holds(p_1, T), \dots, holds(p_n, T), not\ holds(q_1, T), \dots, not\ holds(q_n, T), time(T).$$

VI. REPRESENTATION OF DEONTIC STATEMENTS

We assume that for every norm N of the form (4.5) a unique id id is assigned to it. In order to represent a norm, a rule of the respective type below is made (π_{norm}).

$$\begin{aligned} \text{activating_condition}(id, a, T) \leftarrow & \text{holds}(k_1, T), \dots, \\ & \text{holds}(k_n, T), \\ & \text{not holds}(l_1, T), \dots, \\ & \text{not holds}(l_n, T), \\ & \text{time}(T). \end{aligned}$$

$$\begin{aligned} \text{disact_condition}(id, a, T) \leftarrow & \text{holds}(u_1, T), \dots, \\ & \text{holds}(u_n, T), \\ & \text{not holds}(v_1, T), \dots, \\ & \text{not holds}(v_n, T), \\ & \text{time}(T). \end{aligned}$$

$$\begin{aligned} \text{maintain_cond}(id, a, T) \leftarrow & \text{holds}(w_1, T), \dots, \\ & \text{holds}(w_n, T), \\ & \text{not holds}(z_1, T), \dots, \\ & \text{not holds}(z_n, T), \\ & \text{action}(Action), \text{time}(T). \end{aligned}$$

Keeping in mind the normative semantics described in IV-B we define the rules that implement a norm's lifecycle (π_{lc}).

$$\begin{aligned} \text{obligation_activated}(Id, Action, T) \leftarrow & \\ & \text{activating_condition}(Id, Action, T), \\ & \text{not active}(Id, Action, T - 1), \\ & \text{action}(Action), \text{time}(T), \\ & \text{obligation_id}(Id). \end{aligned}$$

$$\begin{aligned} \text{obligation_disactivated}(Id, Action, T) \leftarrow & \\ & \text{disact_condition}(Id, Action, T), \\ & \text{active}(Id, Action, T - 1), \\ & \text{action}(Action), \text{time}(T), \\ & \text{obligation_id}(Id). \end{aligned}$$

The two rules below define when an obligation is active.

$$\begin{aligned} \text{active}(Id, Action, T) \leftarrow & \\ & \text{obligation_activated}(Id, Action, T), \\ & \text{action}(Action), \text{time}(T), \\ & \text{obligation_id}(Id). \end{aligned}$$

$$\begin{aligned} \text{active}(Id, Action, T) \leftarrow & \text{active}(Id, Action, T - 1), \\ & \text{not obligation_disactivated}(Action, T), \\ & \text{action}(Action), \text{time}(T), \text{obligation_id}(Id). \end{aligned}$$

VII. REPRESENTATION OF PLANNING

We assume that for every action we have an estimated duration on its execution time:

$$\text{estimated_time}(a, t) \leftarrow .$$

where t is a constant.

The generic rule

$$\begin{aligned} \text{started}(Action, T) \leftarrow & \text{obligation_activated}(Id, Action, T), \\ & \text{executable}(Action, T), \text{time}(T), \\ & \text{action}(Action), \text{obligation_id}(Id). \end{aligned}$$

ensures that an action starts whenever it is executable and an obligation to execute it becomes active.

$$\begin{aligned} \text{terminated}(Action, T2) \leftarrow & \text{started}(Action, T1), \\ & \text{estimated_time}(Action, EstimatedTime), \\ & T2 = T1 + EstimatedTime, \\ & \text{time}(T2), \text{time}(T1). \end{aligned}$$

The generic rule

$$\begin{aligned} \text{violation}(Id, Action, T) \leftarrow & \text{active}(Id, Action, T), \\ & \text{not maintain_cond}(Id, Action, T), \\ & \text{action}(Action), \text{time}(T), \text{obligation_id}(Id). \end{aligned}$$

detects a violation whenever a clause is active and the maintenance condition is not true.

The following propositions ensure the correctness of the program π implementing the norm's lifecycle properties.

Proposition 3: Let D be a consistent domain description, O be an initial state complete set of observations such that (D, O) is consistent, DS a set of norms and M a stable model of π . If there is a state σ reached by Φ_D such that $D \models_{DS, O, \sigma} \text{activated}(N)$ then there is a $T \in \mathbb{N}$ such that $\text{activating_condition}(id, a, T) \in M$ where id is the unique identifier of the norm N and a is the *DeonticStatement* of the norm N .

Proposition 4: Let D be a consistent domain description, O be an initial state complete set of observations such that (D, O) is consistent, DS a set of norms and M a stable model of π . If there is a state σ reached by Φ_D such that $D \models_{DS, O, \sigma} \text{disactivated}(N)$ then there is a $T \in \mathbb{N}$ such that $\text{disact_condition}(id, a, T) \in M$ where id is the unique identifier of the norm N and a is the *DeonticStatement* of the norm N .

Proposition 5: Let D be a consistent domain description, O be an initial state complete set of observations such that (D, O) is consistent, DS a set of norms and M a stable model of π . If there is a state σ reached by Φ_D such that $D \models_{DS, O, \sigma} \text{violated}(N)$ then there is a $T \in \mathbb{N}$ such that $\text{maintain_cond}(id, a, T)$ and $\text{active}(id, a, T) \in M$ where

id is the unique identifier of the norm N and a is the *DeonticStatement* of the norm N .

Example 4: Let D , O , DS be the domain description, observations and deontic statements of the examples 1 and 2, π the program modeling D , O and DS and M one of its stable models. As seen in the example 3, in the state $\sigma = [repair(rc)]\sigma_0$ *activated*(N_{RC2}) will hold. Thus, there for some $T \in \mathbb{N}$ *activating_condition*($id_{RC2}, repair(rc), T$) $\in M$ where id_{RC2} is the unique identifier of the norm RC2.

VIII. EXAMPLE

In our example, the initial observation can be:

holds(*at_shop*, 0).

time(0..30).

The model will include the following:

obligation_activated(*rc1*, *repair*(*rc*), 0)
obligation_disactivated(*rc1*, *repair*(*rc*), 10)

obligation_activated(*rc2*, *pay*(*cl*), 10)
obligation_disactivated(*rc2*, *pay*(*cl*), 11)

action_started(*repair*(*rc*), 0)
action_finished(*repair*(*rc*), 10)

action_started(*pay*(*cl*), 10)
action_finished(*pay*(*cl*), 11)

For further information and the ASP code, the reader is referred to the web page where the code is hosted: <http://www.lsi.upc.edu/panagiotidi/aspcode>

IX. CONCLUSIONS, DISCUSSION AND FUTURE WORK

Assuming the environment that an agent operates can be modelled by an action language, norms impose restrictions on the behaviour of the agent. This paper presents ongoing work on the formal specification and development of a framework that enables to model an action language with respect to a normative environment. Semantics of the action language are based on [6] and the language is partially implemented and further enriched with temporal projection. The deontic statements such as obligations and permissions as well as properties related to the norm's lifecycle (activation, disactivation, violation) are formalised and implemented according to [20]. An attempt to define and apply planning techniques within the formal model and the implementation is made.

This piece of work can be extended in various directions. Formal aspects of planning need to be carefully considered

and taken into consideration while advancing with the use of the framework in order to reach conclusions on the future states. A formal notion of goal needs to be introduced in order to reach a desired state. Another interesting aspect to be examined is the use of weighted rules when reasoning with respect to norms against the possible effects of one or more violations. This will ensure that each agent can make decisions on execution time based not only on a set of norms indicating a desired behaviour but also on criteria such as desires, preferences and time constraints.

X. ACKNOWLEDGEMENTS

This research has been funded by the FP7-215890 ALIVE project, funded by the European Commission. Javier Vázquez-Salceda's work has been also partially funded by the "Ramón y Cajal" program of the Spanish Ministry of Education and Science.

REFERENCES

- [1] J. Vázquez-Salceda and F. Dignum, "Modelling electronic organizations," *Multi-Agent Systems and Applications III: 3rd. International/Central and Eastern European Conference on Multi-Agent Systems -CEEMAS'03, Lecture Notes in Artificial Intelligence 2691*, pp. 584–593, 2003. [Online]. Available: <http://people.cs.uu.nl/dignum/papers/harmonia-CEEMAS03.pdf>
- [2] J.-J. C. Meyer and R. J. Wieringa, "Deontic logic: a concise overview," pp. 3–16, 1993.
- [3] H. Aldewereld, *Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols*. PhD thesis, Utrecht University, 2007.
- [4] C. Baral, M. Gelfond, and A. Provetti, "Representing actions: Laws, observations and hypotheses," *The Journal of Logic Programming*, vol. 31, no. 1-3, pp. 201–243, 1997.
- [5] C. Baral and M. Gelfond, "Reasoning agents in dynamic domains," 2000, pp. 257–279.
- [6] M. Gelfond and V. Lifschitz, "Representing action and change by logic programs," *Journal of logic programming*, 1993.
- [7] M. Gelfond and J. Lobo, "Authorization and obligation policies in dynamic systems," *Proceedings of the 24th International Conference on Logic Programming*, pp. 22–36, 2008.
- [8] N. Tinnemeier, M. Dastani, and J.-J. Meyer, "Roles and norms for programming agent organizations," in *AAMAS '09: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2009, pp. 121–128.
- [9] C. Baral, *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge: Cambridge University Press, 2003.
- [10] S. DLV, "Vienna University of Technology," <http://www.dbai.tuwien.ac.at/proj/dlv/>, 1996.
- [11] S. SMOBELS, "Helsinki University of Technology," <http://www.tcs.hut.fi/Software/smodels/>, 1995.
- [12] Potassco, "University of Potsdam," <http://potassco.sourceforge.net/>, 2007.
- [13] T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres, "A logic programming approach to knowledge-state planning, ii: The DLV^k system." *Artif. Intell.*, vol. 144, no. 1-2, pp. 157–211, 2003.
- [14] S. Dworschak, S. Grell, V. Nikiforova, T. Schaub, and J. Selbig, "Modeling Biological Networks by Action Languages via Answer Set Programming," *Constraints*, vol. 13, no. 1, pp. 21–65, 2008.
- [15] C. Baral, K. Chancellor, N. Tran, N. Tran, A. M. Joy, and M. E. Berens, "A knowledge based approach for representing and reasoning about signaling networks," in *ISMB/ECCB (Supplement of Bioinformatics)*, 2004, pp. 15–22.
- [16] J. C. Nieves, M. Osorio, and U. Cortés, "Preferred Extensions as Stable Models," *Theory and Practice of Logic Programming*, vol. 8, no. 4, pp. 527–543, July 2008.

- [17] N. Oren, S. Panagiotidi, J. Vázquez-Salceda, S. Modgil, M. Luck, and S. Miles, "Towards a formalisation of electronic contracting environments," *Coordination, Organization, Institutions and Norms in Agent Systems, the International Workshop at AAAI 2008, pages 61-68, Chicago, Illinois, USA, 2008*.
- [18] R. Confalonieri, S. Alvarez-Napagao, S. Panagiotidi, J. Vázquez-Salceda, and S. Willmott, "A middleware architecture for building contract-aware agent-based services," *LECTURE NOTES IN COMPUTER SCIENCE*, Jan 2008. [Online]. Available: <http://www.springerlink.com/index/m84n55h118j86178.pdf>
- [19] M. Gelfond and V. Lifschitz, "The stable model semantics for logic programming." R. Kowalski and K. Bowen, editors, *5th Conference on Logic Programming, pages 1070-1080. MIT Press, 1988*.
- [20] S. Panagiotidi, J. Vázquez-Salceda, S. Alvarez Napagao, S. Ortega-Martorell, S. Willmott, R. Confalonieri, and P. Storms, "Intelligent contracting agents language," *Proceedings of the Symposium on Behaviour Regulation in Multi-Agent Systems -BRMAS'08-, Aberdeen, UK, April 2008, 1988*.
- [21] G. Boella and L. van der Torre, "Permissions and obligations in hierarchical normative systems," *Proceedings of the 9th international conference on Artificial intelligence and law*, pp. 109-118, 2003.

Languages, methodologies and Development tools for multi-agent systems, LADS'009

(Introductory Essay of the Workshop)

Mehdi Dastani*, Amal El Fallah Seghrouchni†, João Leite‡, and Paolo Torroni§

* Department of Information and Computing Sciences, Utrecht University,

Email: mehdi@cs.uu.nl

† LIP6 - University Pierre and Marie Curie,

Email: Amal.Elfallah@lip6.fr

‡ CENTRIA & DI, FCT, Universidade Nova de Lisboa

Email: jleite@di.fct.ul.pt

§ Dipartimento di Elettronica, Informatica e Sistemistica, Università di Bologna

Email: paolo.torroni@unibo.it

Abstract

LADS'09 aims to offer a rich forum for leading researchers, from both academia and industry, interested in sharing their experiences about the theory and practice of formal approaches, programming languages, tools and techniques that support the development and deployment of multi-agent systems. These are gaining increasing attention in important application areas such as electronic institutions, semantic web, web services, security, grid computing, ambient intelligence, pervasive computing, electronic contracting, among others.

I. INTRODUCTION

These are the pre-proceedings of the second international workshop on languages, methodologies and development tools for multi-agent systems (LADS'009) to be held on 7-11 September 2009 in Torino, Italy. LADS'009 workshop aims to address both theoretical and practical issues related to developing and deploying multi-agent systems. In particular, it will constitute a rich forum where leading researchers from both academia and industry share their experiences on formal approaches, programming languages, methodologies, tools and techniques that support the development and deployment of multi-agent systems. From theoretical point of view, LADS'009 aims to address issues related to theories, models, and approaches that are needed to facilitate the development of multi-agent systems ensuring their predictability and verifications. From practical point of view, LADS'009 aims at stimulating research and discussion on how multi-agent system specifications and designs can be effectively implemented and tested. LADS'009 workshop promises to provide interesting discussion and exchange of ideas concerning theories, methodologies, techniques and principles that are important for multi-agent programming technology. The programme of the workshop consists of four sessions and covers formal models and theories (session 1), development methodologies and tools (session 2), programming languages (sessions 3), and Architectures (session 4). More details on the programme can be found at: <http://centria.di.fct.unl.pt/events/lads009/>.

LADS'009 Programme Co-chairs

Mehdi Dastani
Amal El Fallah Seghrouchni
João Leite
Paolo Torroni

July 26, 2009

II. WORKSHOP COMMITTEES

A. *Workshop Organizers*

Mehdi Dastani	Utrecht University, The Netherlands
Amal El Fallah Seghrouchni	University of Paris VI, France
João Leite	Universidade Nova de Lisboa, Portugal
Paolo Torroni	University of Bologna, Italy

B. *Programme Commitee*

Marco Alberti	New University of Lisbon, Portugal
Natasha Alechina	University of Nottingham, UK
José Júlio Alferes	New University of Lisbon, Portugal
Matteo Baldoni	University of Torino, Italy
Rafael H. Bordini	Universidade Federal do Rio Grande do Sul, Brazil
Juan Botía	Murcia University, Spain
Lars Braubach	University of Hamburg, Germany
Keith Clark	Imperial College London, UK
Mehdi Dastani	Utrecht University, The Netherlands
Yves Demazeau	Institut IMAG, Grenoble, France
Juergen Dix	Clausthal University, Germany
Amal El Fallah Seghrouchni	University of Paris 6, France
Paolo Giorgini	University of Trento, Italy
Jorge Gómez-Sanz	Universidad Complutense Madrid, Spain
Koen Hindriks	Delft University, The Netherlands
Shinichi Honiden	NII, Tokyo, Japan
Jomi Fred Hübner	Universidade Regional de Blumenau, Brazil
Wojciech Jamroga	University of Luxembourg, Luxembourg
Peep Kúngas	SOA Trader, Ltd., Tallin, Estonia
Joo Leite	New University of Lisbon, Portugal
John Lloyd	Australian National University, Canberra, Australia
Alessio Lomuscio	Imperial College London, UK
Viviana Mascardi	University of Genova, Italy
John-Jules Meyer	Utrecht University, The Netherlands
Alexander Pokahr	University of Hamburg, Germany
Patrick Tallibert	Thales Airborne Systems, Elancourt, France
Paolo Torroni	University of Bologna, Italy
Birna van Riemsdijk	Delft University, The Netherlands
Leon van der Torre	University of Luxembourg, Luxembourg
Gerhard Weiss	Software Competence Center Hagenberg, Austria
Pinar Yolum	Bogazici University, Istanbul, Turkey
Yingqian Zhang	Delft University, The Netherlands

C. *Additional Reviewers*

Alejandro Guerra Hernandez, Matthias Nickles, Akin Gunay, Yasuyuki Tahara, Ozgur Kafali, Mario Mendes

III. LIST OF PAPERS

- Detecting Exceptions in Commitment Protocols: Discovering Hidden States
by *Ozgur Kafali and Pinar Yolum*
- A Methodology for Developing Self-Explaining Agents for Virtual Training
by *Maaïke Harbers, Karel Van den Bosch, and John-Jules Meyer*
- An Integrated Semantics of Social Commitments and Associated Operations
by *Jamal Bentahar, M El-Menshawy, and R. Dssouli*
- Externalisation and Internalization: A New Perspective on Agent Modularisation in Multi-Agent Systems Programming
by *Alessandro Ricci, Michele Piunti, and Mirko Viroli*
- Call Graph Profiling for Multi Agent Systems
by *Dinh Doan Van Bien, David Lillis, and Rem W. Collier*
- ReSeagent: A Refactoring Tool for Plan Level Refactoring in MAS Development
by *Ali Murat Tiryaki and Oguz Dikenelli*
- Programming social middlewares through social interaction types
by *Juan Manuel Serrano and Sergio Saugar*
- Temporal Planning in Dynamic Environments for P-CLAIM Agents
by *Adnan Hashmi and Amal El Fallah Seghrouchni*
- Agents Secure Interaction in Data driven Languages
by *Mahdi Zargayouna, Balbo Flavien, and Serge Haddad*
- Executing Agent Plans by Reducing to Workflows
by *Tayfun Gokmen Halac, Ovunc CETIN, Erdem Eser Ekinici, R. Cenk Erdur, and oguz dikenelli*
- The ARTS Real-Time Agent Architecture
by *Konstantin Vikhorev, Natasha Alechina, and Brian Logan*

IV. ACKNOWLEDGMENT

The co-chairs of this workshop would like to thank all authors, invited speakers, programme committee members, and additional reviewers for their outstanding contribution to the success of LADS'009. The co-chairs would also like to thank all the sponsors and Springer. We are particularly grateful to MALLOW'009 organisers, Matteo Baldoni, Cristina Baroglio, and Guido Boella, for their technical support and for hosting LADS'009.

Detecting Exceptions in Commitment Protocols: Discovering Hidden States

Özgür Kafalı

Department of Computer Engineering
Boğaziçi University
TR-34342, Bebek, İstanbul, Turkey
e-mail: ozgurkafali@gmail.com

Pınar Yolum

Department of Computer Engineering
Boğaziçi University
TR-34342, Bebek, İstanbul, Turkey
e-mail: pinar.yolum@boun.edu.tr

Abstract—Open multiagent systems consist of autonomous agents that are built by different vendors. In principle, open multiagent systems cannot provide any guarantees about the behaviors of their agents. This means that when agents are working together, such as carrying out a business protocol, one agent's misbehavior may potentially create an exception for another agent and obstruct its proper working. Faced with such an exception, an agent should be able to identify the problem by verifying the compliance of other agents.

Previous work on verification of protocols unrealistically assume that participants have full knowledge of a protocol. However, when multiple agents enact a protocol, each agent has access to its part of the protocol and not more. This will require agents to check verification by querying others and more importantly by discovering the contracts between them. Here, we propose a commitment-based framework for detecting exceptions in which an agent augments its part of the protocol with its knowledge to construct states that are previously hidden to the agent by generating possible commitments between other agents. The agent then queries others to confirm those states. Our framework is formalized using C+ and is tested using a realistic business scenario.

I. INTRODUCTION

In open multiagent systems, it is possible for agents to interact with others that they have no previous knowledge of. Carrying out interactions, such as business dealings, with such others is difficult since there is no guarantee about how the other agents will act. If others do not follow their parts of the interactions, the entire business may be jeopardized. This requires an agent participating in such a situation to be able to verify that others are acting by the rules.

Verification is especially important in the face of *exceptions*. Here, we deal with high-level exceptions that pertain to the workings of the underlying protocol. For example, if a buyer does not receive a merchandise that was scheduled for delivery, it can conclude that there must have been an exception in the workings of the entire protocol. When such an exception occurs, the agent facing the exception needs to identify the problem behind it. This is a two-phase procedure; first detecting the exception, and then taking proper action recover from the exceptional situation. In this paper, we focus on the first phase. That is, we propose an algorithm for finding the source of exceptions (i.e., caused by which parties and why). In addition, if the source of the exception is identified

correctly, then it is a forward step in the recovery process, because the agent facing the exception has a means of proof for the cause of it. This proof can then be used to consult other authorities, which can resolve inconsistencies between parties.

Realistic business affairs consist of multiple parties that carry out different tasks. Multiparty interactions have two inherent properties; (1) interactions between different parties are regulated by different contracts (i.e., a seller may exercise different rules when dealing with an individual versus a corporation), (2) rules of interaction between different parties are private and not revealed to the outside world (i.e., a contract between a seller and a carrier may never be revealed to buyers publicly). While these properties are essential for multiparty protocols, they pose important challenges for verification, which brings the question of how an agent can verify others' compliance when it has only partial information about their activities. We use the scenario in Example 1 throughout the paper as our running example.

Example 1. Consider the simple purchase-and-delivery protocol that includes three business roles. The roles in the protocol are *customer*, *bookstore*, and *deliverer*. In a normal execution, the customer buys a book from the bookstore and the deliverer delivers the book to the customer. However, certain exceptions may occur during the enactment of this protocol. For example, consider the case where the customer pays for the book and expects delivery in three days. In addition, suppose that the bookstore sends books to be delivered to the deliverer in large groups. If at the time the customer buys the book, the number of books pending for delivery at the bookstore is not enough, the book will not be delivered causing an exception for the customer. However, since the customer does not know the details of the contract between the bookstore and the deliverer, the source of the exception is not immediately clear to the customer. One option for the customer is to simply ask the bookstore about the cause of exception. However, this may not be possible in some situations (i.e., the bookstore is not willing to share information regarding its contracts with other parties, or the exception is caused by a party beyond the knowledge of the bookstore). Then, the customer has to use its knowledge first to predict possible causes, and query corresponding agents

to determine which one is the actual cause of the exception.

In order to study verification rigorously, we capture agents' interactions through commitments [1], and adopt C+ as a language to formalize those interactions [2], [3]. In contrast to previous work on verification, we propose a realistic exception discovery framework in which; (1) multiple roles exist in the business, (2) business scenarios are distributed (each role has its own view of the protocol), and (3) each agent deals with an exception by discovering contracts of other agents. With this proposed approach, an agent only finds out the necessary details to continue its operation in tracing down the in-compliant agents.

The rest of this paper is organized as follows. Section II gives necessary background on protocols, commitments and C+. Section III describes the running example and defines the problem formally. Section IV introduces our solution to deal with exceptions in distributed scenarios, and Section V explains its details. Section VI presents a discussion of our work with comparisons to the literature and provides directions for further research.

II. TECHNICAL BACKGROUND

In this section, we first describe formally what we mean by a business protocol, then we review the necessary concepts related to specifying commitments, and realizing them in a formal description language.

A. Protocols & Runs

Definition II.1. A protocol \mathcal{P} is a 6-tuple $\langle \mathbb{S}, \mathbb{A}, \mathbb{C}, \mathbb{R}, \mathcal{S}_{\mathcal{I}}, \mathcal{S}_{\mathcal{F}} \rangle$, such that \mathbb{S} is a finite set of states, \mathbb{A} is a finite set of actions, \mathbb{C} is a finite set of conditions, \mathbb{R} is a finite set of roles, $\mathcal{S}_{\mathcal{I}}$ is the set of initial states ($\mathcal{S}_{\mathcal{I}} \subset \mathbb{S}$), and $\mathcal{S}_{\mathcal{F}}$ is the set of final states ($\mathcal{S}_{\mathcal{F}} \subset \mathbb{S}$). Intermediate (middle) states $\mathcal{S}_{\mathcal{M}}$ are states that are not in $\mathcal{S}_{\mathcal{I}}$ or in $\mathcal{S}_{\mathcal{F}}$.

Definition II.2. A state is a set of conditions and commitments that hold in it.

Definition II.3. A run \mathcal{R} of a protocol \mathcal{P} is simply a sequence of states $\langle S_0, \dots, S_n \rangle$ starting from an initial state ($S_0 \in \mathcal{S}_{\mathcal{I}}$). For now, we consider only finite runs.

Definition II.4. A desirable run is the one that ends in a final state ($S_n \in \mathcal{S}_{\mathcal{F}}$).

Definition II.5. An exceptional run is the one that ends in an intermediate state ($S_n \in \mathcal{S}_{\mathcal{M}}$), and thus does not reach a final state.

Desirable runs are preferred by agents since they lead them to reach their goals, whereas exceptional runs are unexpected by agents and proper action (i.e., exception handling routines) has to be taken in order for the protocol to evolve from those states.

Definition II.6. An agent-centric sub-protocol $\mathcal{P}' \langle \mathbb{S}', \mathbb{A}', \mathbb{C}', \mathbb{R}', \mathcal{S}'_{\mathcal{I}}, \mathcal{S}'_{\mathcal{F}} \rangle$ is a subset of the main protocol $\mathcal{P} \langle \mathbb{S}, \mathbb{A}, \mathbb{C}, \mathbb{R}, \mathcal{S}_{\mathcal{I}}, \mathcal{S}_{\mathcal{F}} \rangle$ in which; (1) $\forall s' \in \mathbb{S}', \exists s \in \mathbb{S} : s' \subseteq s$,

(2) $\mathbb{A}' \subseteq \mathbb{A}$, (3) $\mathbb{C}' \subseteq \mathbb{C}$, (4) $\mathbb{R}' \subseteq \mathbb{R}$, (5) $\forall s' \in \mathcal{S}'_{\mathcal{I}}, \exists s \in \mathcal{S}_{\mathcal{I}} : s' \subseteq s$, (6) $\forall s' \in \mathcal{S}'_{\mathcal{F}}, \exists s \in \mathcal{S}_{\mathcal{F}} : s' \subseteq s$.

B. Commitments

Commitments are formed between two agents and roughly correspond to obligations [1]. The debtor of a commitment is the agent that is committed to bring about a condition. The creditor benefits from the commitment. Commitments are created and discharged by the interactions of the agents. There are two types of commitments:

c(x, y, p): This is a base-level commitment between debtor x and creditor y with proposition p . When this commitment is in charge, debtor x becomes committed to creditor y for satisfying p .

cc(x, y, p, q): This is a conditional commitment between debtor x and creditor y with condition p and proposition q . When this commitment is in charge, if p is satisfied (by y), x will become committed to y for satisfying q .

The following four operations describe how commitments are manipulated throughout a protocol. We assume that each protocol action initiates a commitment operation (i.e., altering a contract between agents). Thus, commitment operations describe the semantics of protocol actions.

create(x, c(x, y, p)): This operation initiates the creation of the base-level commitment c . It is performed by x , the debtor of the commitment. Since this operation creates a new commitment which does not hold previously, it causes a state transition ($S_i \xrightarrow{\text{create}(x, c(x, y, p))} S_i \cup \{c(x, y, p)\}$).

ccreate(x, cc(x, y, p, q)): This operation initiates the creation of the conditional commitment cc . It is performed by x , the debtor of the commitment. This operation also causes a state transition ($S_i \xrightarrow{\text{ccreate}(x, cc(x, y, p, q))} S_i \cup \{cc(x, y, p, q)\}$).

discharge(x, c(x, y, p)): This operation resolves the base-level commitment c . It is performed by x , the debtor of the commitment, and the commitment c is terminated afterward. A base-level commitment is resolved when the proposition p of the commitment becomes true. This operation causes a state transition since a previously holding commitment disappears ($S_i \xrightarrow{\text{discharge}(x, c(x, y, p))} S_i - \{c(x, y, p)\} \cup \{p\}$).

cdischarge(x, cc(x, y, p, q)): This operation resolves the conditional commitment cc . It is performed by x , the debtor of the commitment, and the conditional commitment cc is terminated afterward. If the proposition q of a conditional commitment cc becomes true, then cc is discharged immediately causing a state transition ($S_i \xrightarrow{\text{cdischarge}(x, cc(x, y, p, q))} S_i - \{cc(x, y, p, q)\} \cup \{q\}$). If the condition p of cc is brought about, then cc is discharged, and a new base-level commitment is created with the proposition q of cc causing another state transition ($S_i \xrightarrow{\text{cdischarge}(x, cc(x, y, p, q))} S_i - \{cc(x, y, p, q)\} \cup \{c(x, y, q)\} \cup \{p\}$).

C. Commitment Protocols

In this section, we integrate commitments into protocols. Definitions II.7, II.8, and II.9 provide useful properties re-

garding states with respect to commitments.

Definition II.7. A state is inconsistent if it is one of the following; (1) state $S_i = \{cc(x, y, p, q), p\}$ is inconsistent since the conditional commitment cannot coexist with its condition, (2) state $S_j = \{p, \neg p\}$ is inconsistent since a condition cannot coexist with its negation.

Definition II.8. Two states are equivalent with respect to an agent if they share the same conditions and commitments regarding that agent.

Example 2. Let $S_i = \{cc(x, y, p, q), r\}$ and $S_j = \{cc(x, y, p, q), cc(y, z, v, w), r, u\}$ be two states, and assume r is a condition that agent x can bring about (but does not affect agent y 's working) and u is a condition that agent y can bring about (but does not affect agent x 's working). Then, S_i and S_j are equivalent states for agent x (since the commitment $cc(y, z, v, w)$ is irrelevant to agent x), but not equivalent states for agent y (since the commitment $cc(x, y, p, q)$ that is related to agent y does not hold in state S_i , but holds in state S_j).

Definition II.9. The distance between two states S_i and S_j is the number of commitment operations that are required to bring the protocol from state S_i to state S_j .

Example 3. Let $S_i = \{cc(x, y, p, q)\}$ and $S_j = \{c(x, y, q), cc(y, z, q, r), p\}$ be two states. Then, the distance between states S_i and S_j is 2 since it takes two commitment operations to go from state S_i to S_j ; a *ccreate* operation to create $cc(y, z, q, r)$, and a *cdischarge* operation to resolve $cc(x, y, p, q)$ into $c(x, y, q)$.

D. The Action Description Language C+

We realize the business scenarios to be described using commitment protocols specified in the action description language, C+ [2], [3]. A protocol in C+ is composed of a set of states and transitions between states (i.e., a transition system). A state may contain several fluents that hold in that state (true propositions). A fluent's value is changed as the consequence of an action that is performed by an agent. An inertial fluent is the one whose value is not changed until an action makes it change. Our use of C+ for formalizing commitments and their operations are based on that of Chopra and Singh [3].

Listing 1 shows how commitment operations are realized in C+. This is a basis for other protocol specifications that utilize commitments. Through lines 10-14, commitments and conditional commitments are modeled as inertial fluents. Commitment operations shown through lines 17-22 are modeled as auxiliary (i.e., simple) actions. An auxiliary action has to be initiated by a protocol action and cannot be performed independently. The causation rules associated with those operations are shown through lines 25-32.

III. BUSINESS SCENARIO

In order to show how commitments are utilized in real business environments, we describe in detail our running example that represents concrete business interactions. Figure 1 describes the purchase protocol introduced in Section I. There

are four states (S_0, S_1, S_2 , and S_3), three actions that enable the transitions between the states (*sendPayment*, *sellBook*, and *deliverBook*), and three conditions corresponding to the outcomes of the actions in the protocol (*payc*, *bookc*, and *deliverc*). There is a single initial state (S_0), a single final state (S_3), and two intermediate states (S_1 and S_2).

```

:- sorts
   role;
   condition .
1
2
3

:- variables
   x, y, z :: role;
   p, q :: condition .
4
5
6
7

% Declaration of commitments
:- constants
   commitment(role, role, condition)
   :: inertialFluent;
   ccommitment(role, role, condition, condition)
   :: inertialFluent;
8
9
10
11
12
13
14

% Commitment operations
create(role, role, condition) :: action;
15
16
17
18
19
20
21
22
discharge(role, role, condition) :: action;
ccreate(role, role, condition, condition)
:: action;
cdischarge(role, role, condition, condition)
:: action;

% Commitment rules
23
24
create(x,y,p) causes commitment(x,y,p)
where x > y .
25
26
discharge(x,y,p) causes -commitment(x,y,p)
where x > y .
27
28
ccreate(x,y,p,q) causes ccommitment(x,y,p,q)
where x > y & p > q .
29
30
cdischarge(x,y,p,q) causes -ccommitment(x,y,p,q)
& commitment(x,y,q) where x > y & p > q .
31
32

```

Listing 1. Commitment Operations in C+

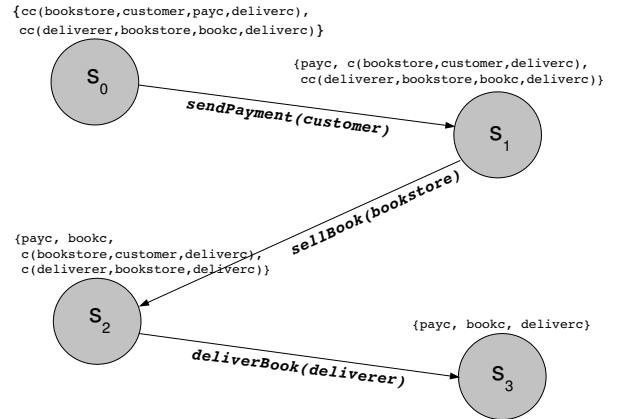


Fig. 1. Purchase & Delivery Protocol

No conditions initially hold in S_0 , but two conditional commitments are present. The first commitment $cc(\text{bookstore}, \text{customer}, \text{payc}, \text{deliverc})$ means that the bookstore is committed to make sure that the book is delivered if the customer pays for it. The second commitment $cc(\text{deliverer}, \text{bookstore},$

$bookc, deliverc$) means that the deliverer is committed to deliver the book to the customer if the bookstore sends it. Since the customer's goal is to get the book delivered, it performs the $sendPayment$ action. This brings the protocol to state S_1 where condition $payc$ holds as the outcome of the $sendPayment$ action. Also, the conditional commitment $cc(bookstore, customer, payc, deliverc)$ is discharged to the base-level commitment $c(bookstore, customer, deliverc)$. Next, the bookstore performs the $sellBook$ action which brings the protocol to state S_2 . Accordingly, condition $bookc$ holds and the conditional commitment $c(deliverer, bookstore, bookc, deliverc)$ is discharged to the base-level commitment $c(deliverer, bookstore, deliverc)$. Finally, the deliverer performs the $deliverBook$ action which brings the protocol to state S_3 . Both commitments in S_2 are discharged and condition $deliverc$ holds in S_3 . State S_3 is the final state for the protocol since all three conditions hold at the same time. Thus, a desirable run for the protocol is $\langle S_0, S_1, S_2, S_3 \rangle$.

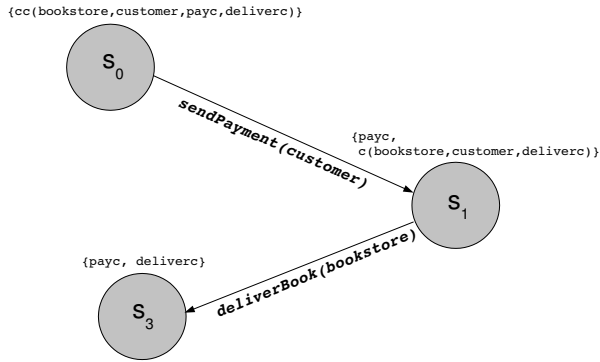


Fig. 2. $\mathcal{P}_{Customer}$ Sub-Protocol

Note that states S_1 and S_2 are equivalent states for the customer, because condition $payc$ and commitment $c(bookstore, customer, deliverc)$ hold in both states. In addition, condition $bookc$ and the discharged commitment $cc(deliverer, bookstore, bookc, deliverc)$ are irrelevant to the customer. Thus, the customer's sub-protocol $\mathcal{P}_{Customer}$ includes states S_0 , S_1 , and S_3 as shown in Figure 2. S_0 is the initial state, S_3 is the final state, and S_1 is the only intermediate state for this sub-protocol. There are two actions ($sendPayment$ and $deliverBook$) and two conditions ($payc$ and $deliverc$).

Listing 2 describes part of the customer's protocol in C+. Line 2 includes the commitment operations as introduced in Listing 1. Lines 4-6 define the roles and conditions that are involved in the protocol. Lines 9-11 define the fluents representing the messages that hold in certain states of the protocol. For example, the message $pay(customer, bookstore)$ has the meaning that the customer has paid the bookstore for the book. The fluents in line 12 define the initial and final conditions for the protocol. The protocol actions are defined through

lines 15-17. The $initiate$ action is performed by the role $super$ to initialize the conditional commitments between the parties (i.e., $super$ can be considered as a protocol designer). Certain actions cannot be performed by some agents. As line 20 suggests, the $sendPayment$ action cannot be performed by the bookstore.

```

1 % Include the commitment operations
2 :-include 'com-spec'.
3
4 :- objects
5   super, customer, bookstore, deliverer :: role;
6   payc, bookc, deliverc :: condition.
7
8 % Fluents that define the states of the protocol
9 :- constants
10  init(role), pay(role, role), book(role, role),
11  deliver(role, role) :: inertialFluent;
12  initial, final :: sdFluent.
13
14 % Protocol actions
15 initiate(role), sendPayment(role),
16 sellBook(role), deliverBook(role)
17 :: exogenousAction;
18
19 % Certain actions are done by specific roles only
20 nonexecutable sendPayment(bookstore).
21 ...
22
23 % Protocol action sendPayment is visible to the
24 % customer agent
25 sendPayment(customer) causes
26   pay(customer, bookstore) if
27   ccommitment(bookstore, customer, payc, deliverc).
28 sendPayment(customer) causes
29   discharge(customer, bookstore, payc)
30   if commitment(customer, bookstore, payc).
31 sendPayment(customer) causes
32   cdischarge(bookstore, customer, payc, deliverc) if
33   ccommitment(bookstore, customer, payc, deliverc).
34 nonexecutable sendPayment(customer)
35   if pay(customer, bookstore) ++ -init(super).
36
37 % Other protocol actions are not visible to the
38 % customer agent
39 ...
40
41 % Causation relations for initial and final states
42 caused initial if initial.
43 caused -initial if pay(x,y).
44 ...
45 caused -initial if ccommitment(x,y,p,q).
46 caused -final if -final.
47
48 % In final state, if pay, book, and deliver holds
49 caused final if pay(customer, bookstore) &
50   book(bookstore, deliverer) &
51   deliver(deliverer, customer).
52 ...
    
```

Listing 2. Customer's Protocol Described in C+

The rules for the protocol action $sendPayment$ are given through lines 25-35. The first rule tells that the fluent $pay(customer, bookstore)$ will start to hold as a result of the protocol action $sendPayment(customer)$ if the conditional commitment $cc(bookstore, customer, payc, deliverc)$ exists prior to it (lines 25-27). The next two rules through lines 28-33 describe how existing commitments are resolved and new commitments are created as a result of the same action. The

last rule ensures that the action $sendPayment(customer)$ is not performed if the payment is already made by the customer or the protocol is not initialized yet by $super$ (lines 34-35). Since the scenario is distributed, other protocol actions, such as $sellBook$ or $deliverBook$, are not accessible (i.e., hidden) by the customer agent. The protocol starts with the state $initial$ and is expected to terminate in state $final$ (lines 42-46), if the required fluents hold (lines 49-51).

Now, let us study what can go wrong in a given protocol run and what exceptions can take place. If an expected action is not performed by an agent that is responsible for performing it, an exception occurs. Two such exceptional runs for this protocol are $\langle S_0, S_1 \rangle$ and $\langle S_0, S_1, S_2 \rangle$. The former run gets stuck at state S_1 , because the bookstore does not send the book to the deliverer. The latter run gets stuck at state S_2 , because the deliverer does not deliver the book to the customer. When one of these exceptions occur, the customer agent cannot find its cause immediately (i.e., in which of the main protocol states the run gets stuck) since states S_1 and S_2 are equivalent for it. However, in order for the customer to deal with the exception, it is crucial that it learns about which agent is causing the exception. Next, we look at the general idea behind our proposed solution, and then explain the details of our approach.

IV. PROPOSED SOLUTION

When faced with an exception, an agent tries to figure out what might have gone wrong. Figure 3 summarizes the approach that agents utilize when detecting exceptions. First, the agent reasons using its own knowledge-base. In many cases, this would not be enough to identify the exception. However, in many settings, as time evolves, new information about the environment becomes available (step 1). Based on the new information, the agent again tries to predict possible contracts between other agents so that it can figure out what has been violated to cause an exception (step 2). Once the agent has possible ideas about what might have gone wrong, it queries other agents that are related to the possible cause of the exception and asks them to confirm one of the possibilities (step 3).

For the above example, this would work as follows: At the beginning, note that the customer is not aware of the existence of a deliverer since its sub-protocol does not include such a role. Thus, its knowledge base includes only the bookstore other than itself. In addition, the conditions initially known by the agent are limited to $payc$ and its goal condition $deliverc$. With this information only, it is not possible to construct state S_2 since it involves a commitment between the bookstore and the deliverer. However, even if its knowledge base does not contain that information, the customer agent may become aware of other roles, and extend its sub-protocol with new information revealed by other agents. For example, if the bookstore announces that the book is sent to the deliverer, then the customer will be aware of the existence of a deliverer role and the condition $bookc$. Information exposure is a simple task that is often performed in real-life delivery scenarios. Now, the

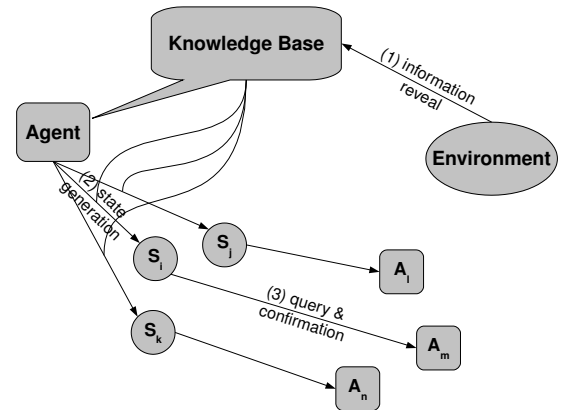


Fig. 3. General Approach

agent has enough knowledge to generate other possible states of the protocol. Once the states are generated, they need to be verified to find out whether they have caused the exception. Accordingly, the agent directs the query about each generated state to one of the agents related to that state (i.e., involved in a commitment within that state).

V. IDENTIFYING EXCEPTION SOURCES

When an exception takes place, it is necessary for the agent to identify who caused the exception so that it can deal with the exception accordingly. As seen in the previous section, this is not easy since an agent may view a number of states identical when indeed they are different for other agents. The question then is how can an agent construct possible real states of the world? If the agent can generate such possibilities, then it can query the involved agents and ask them to confirm one of these states. Next, we present such an algorithm. Without loss of generality, we assume that the algorithm is used by the *customer* agent.

A. State Prediction Algorithm

In this section, we propose an algorithm for the agents to use for constructing the hidden states (i.e., unknown states prior to exception) that might be the cause of exceptions. In order to construct a state, the agent has to generate the possible conditions and commitments that hold in that state. Recall that each agent is only aware of the commitments it is involved in. So, the agent has to predict the possible commitments among other participants to fill the definition of a hidden state.

Algorithm 1 describes how the agent predicts the hidden states for detecting exceptions. The requirements for the algorithm to execute properly are; current state of the agent in its sub-protocol, its goal condition, commitments it is involved in, conditions and roles it is aware of, and a maximum allowed distance parameter for selecting states to query. The algorithm consists of two stages; state generation and state selection, that we describe next.

State Generation Stage: This stage starts with creating a set

Algorithm 1 Predicting Hidden States

Require: S_c {current state}
Require: C_{goal} {goal condition}
Require: $commitments$ {initial commitments}
Require: \mathbb{C} {conditions whose existence are known}
Require: \mathbb{R} {roles whose existence are known}
Require: $dist$ {maximum allowed distance}

{I. State Generation Stage}

- 1: $\mathbb{S} \leftarrow \{S_c\}$ {add current state to the generated states}
- 2: **for all** $commitment_i$ in $commitments$ **do**
- 3: $S \leftarrow S_c$ {create a new state from current state}
- 4: $cc \leftarrow cc(GoalRole, CondRole, Cond, Goal)$
- 5: $Goal \leftarrow C_{goal}$ {replace goal condition}
- 6: $GoalRole \leftarrow select(\mathbb{R})$ {pick a role}
- 7: generate $Cond$ and $CondRole$ using $commitment_i$
- 8: $cond \leftarrow select(\mathbb{C})$ {pick a set of conditions}
- 9: apply commitment operations on cc assuming conditions in $cond$ holds
- 10: $S \leftarrow S \cup cc$ {add the commitments to the state}
- 11: $S \leftarrow S \cup cond$ {add the set of conditions to the state}
- 12: $\mathbb{S} \leftarrow \mathbb{S} \cup \{S\}$ {add the generated state to the result}
- 13: **end for**

{II. State Selection Stage}

- 14: **for all** S_i in \mathbb{S} **do**
- 15: **if** $distance(S_i, S_c) > dist$ **then**
- 16: $\mathbb{S} \leftarrow \mathbb{S} - \{S_i\}$ {remove the state from the result}
- 17: **end if**
- 18: **end for**
- 19: **return** \mathbb{S}

for storing generated states and the current state of the agent is added to this set (line 1). A generated state is not constructed from scratch, but rather extended from the current state of the agent (line 3). In order to fill the state definition, the agent generates the hidden commitments between other parties starting with a conditional commitment template with two roles and two conditions (line 4). The goal condition for the commitment ($Goal$) is the agent's goal (line 5), and the business party that can bring about that condition ($GoalRole$) is picked from the set of roles the agent is aware of (line 6). In order to fill the middle parts of the commitment ($Cond$ and $CondRole$), the agent traces through its own commitments and finds which parties it has a commitment with. For each commitment $cc(x, Role, Cond, p)$ or $cc(Role, x, p, Cond)$, where $Role$ is the agent's role and $Cond$ is one of the conditions that the agent can bring about, it replaces $CondRole$ and $Cond$ of the template commitment using all possible pairs of x and p as line 7 suggests. The agent then searches for conditions to put into the state definition (line 8). Those conditions are also used in applying commitment operations on the generated commitments. Since the generated commitments are the initial versions of contracts between other parties, they might have been changed during the execution of the protocol. Line 9 of the algorithm provides this commitment manipulation

process. Note that no inconsistent states are generated at this stage of the algorithm, because this process resolves necessary commitments with conditions whenever is possible. The state is then ready to be extended with the generated commitments and conditions (lines 10-11). Finally, the state is added to the set of generated states (line 12). This stage continues until no new states are generated.

State Selection Stage: This stage eliminates states generated by the first stage of the algorithm which are at a distance from the current state of the agent's sub-protocol. We apply the state distance property to compute the distance value. The maximum allowed distance for selection is a configurable parameter of the algorithm controlled by the $dist$ value in line 15. The number of states selected out of this stage is expected to decrease if we select this parameter to be low. However, it increases the chance that the actual exceptional state is also eliminated by this process.

Example 4. Let us now depict the algorithm using our scenario. Recall that we've considered two exceptional situations; one gets stuck at state S_1 , and the other gets stuck at state S_2 . However, since S_2 is a hidden state for the customer agent, both S_1 and S_2 converge to state S_1 of the customer agent's sub-protocol. At this point, the customer agent thinks that the exception is caused by the bookstore since the delivery will be done by the bookstore according to its sub-protocol. But, suppose that the bookstore agent informs the customer agent on the delivery process. That is, it tells that the book is given to the deliverer agent. Now, the customer agent has extra knowledge with which it can extend its sub-protocol. Now, the customer agent can initiate the state generation process. The goal of the agent is to successfully generate state S_2 and query agents related to that state (i.e., deliverer) to see whether the main protocol is actually in that state. If so, the exception is caused by the deliverer agent, otherwise the bookstore agent is the cause of the exception. Now, suppose the agent has generated several different states among which one of them is the state S_2 . To learn whether the main protocol is in state S_2 , the customer agent queries the deliverer agent to confirm the existence of state S_2 .

B. Implementation & Evaluation

In order to implement our approach, we used C+ to describe the scenario formally as shown partly in Listings 1 and 2, then implemented the state prediction algorithm in Java. In the trivial cases where the initial commitments between the parties are in force, the protocols terminate as desired for the customer agent. However, since our aim is to observe exceptional situations, we disrupt the C+ descriptions of the scenarios (i.e., remove certain commitments) to enable the occurrence of such exceptions. Once certain parts of the scenario descriptions are extracted, the prediction algorithm is run to generate the possible missing states. Finally, one or more generations complete the scenario descriptions as they should be, leading to a desirable run for the customer agent.

The algorithm can be extended to support sequential protocols that involve more than one agent between the initiator (i.e., customer) and the terminator of the protocol (i.e., deliverer). For example, consider an extension to our scenario where books are packaged before they are sent for delivery. This packaging process needs another role, the *packager*, to be present in the protocol. Thus, the algorithm has to generate two conditional commitments instead of one for each state it will generate, involving the contracts between; (1) the bookstore and the packager, and (2) the packager and the deliverer. Our current system supports these extensions.

Correctness of the Algorithm: Here, we discuss the two stages of the algorithm (state generation and state selection) in order to argue on the correctness of our algorithm. That is, the state causing the exception has to be generated in the state generation stage, and it has to be selected as a candidate for querying in the state selection stage. Next, we consider each stage separately:

State Generation Stage: The number of states generated is limited to the knowledge of the agent about the protocol (i.e., the roles and conditions).

Lemma V.1. *Let S_e be a state in protocol P , and let c be a customer agent executing in P . Assume c is currently in state S_c of its sub-protocol P_c ($P_c \subseteq P$), and assume state S_e differs from state S_c in terms of the set of conditions E_{cnd} and the set of commitments E_{cmm} . Let the commitments in E_{cmm} include the set of conditions E_{con} and the set of roles E_{role} , and let $E_{cond} = E_{cnd} \cup E_{con}$. Now, if c faces an exception caused at state S_e , and if c knows about the conditions in E_{cond} and the roles in E_{role} , then state S_e will be generated by agent c .*

Proof: Recall that agent c generates a state by filling its definition with conditions and commitments. Agent c tries all possible combinations of conditions and commitments it can generate using its knowledge about P_c . Note that $S_e = S_c \cup E_{cnd} \cup E_{cmm}$, thus agent c needs to generate the conditions in E_{cnd} and the commitments in E_{cmm} . Using its knowledge about E_{cond} , agent c can generate the conditions in E_{cnd} , since $E_{cnd} \subseteq E_{cond}$. Using its knowledge about E_{cond} and E_{role} , agent c can generate the commitments in E_{cmm} , since those commitments are composed of the conditions in E_{con} and the roles in E_{role} which are known by agent c ($E_{con} \subseteq E_{cond}$). Thus, agent c generates state S_e . ■

State Selection Stage: The chance of the exceptional state being selected is related to how distant it is from the current state of the agent's sub-protocol, and the choice of the maximum allowed distance parameter used for deciding whether two states are distant.

VI. DISCUSSION

Commitment protocols have been used before to formalize business scenarios [4], [5]. Chopra and Singh explain how transformation specifications are used in order to extend protocols to cover new situations [3]. Their formalization of

commitment protocols in C+ form the basis of our work. However, Chopra and Singh do not provide mechanisms for distributed verification as we have done here.

Mallya and Singh divide exceptions into two categories [6]; (1) expected exceptions which are handled at design-time using preferences over protocol runs, and (2) unexpected exceptions which occur at run-time and are handled via splicing exception handlers with base protocols. Their work helps protocol designers for handling exceptions. However, handling unexpected exceptions with such generic handlers is costly. The work of Venkatraman and Singh resembles our work since each agent checks compliance on its own [7]. The process is distributed in a sense that each agent has access to its own set of messages during execution, but their business scenario does not fully simulate a distributed environment. Our work differs from theirs since an agent in our scenario needs extra information when resolving an exception.

Our work can also be considered in the multiagent plan execution context for identifying failures. Jonge *et al.* [8] classify the diagnosis of plan failures into two categories; (1) primary plan diagnosis simply points out the failed action, (2) secondary plan diagnosis identifies the actual cause of the failure as we also focus in our work. Although the agents in their plan execution system have partial observations over the system, they still have a major knowledge about their environment. Thus, our work differs from theirs in terms of the distributed protocol execution.

Expectations have also been used to formalize business protocols as described in the SCIFF framework [9]. SCIFF is based on abductive logic, and it does not only specify a business protocol, but also helps verify agent interactions in open systems. Compliance verification has been considered in other domains; in composite Web services [10], or in agent communication languages (ACLs) [11]. An ACL is part of an agent communication framework. The proposed verification process in Guerin and Pitt's work [11] may require access to agent's internal process, whereas our idea of verification depends only on interaction.

Our approach is based on constructing possible hidden states and querying other agents for confirming those states (i.e., identifying which one of them caused the exception in the main protocol). Intuitively, it is reasonable for the agent to query other agents which are committed to it (i.e., the bookstore agent in our example). At this point, we assume that the agent receives honest responses from others. In a real-life scenario, this querying process will continue as a delegation among the agents regarding their commitments (i.e., the bookstore agent redirects the query of the customer agent to the deliverer agent for inspecting the exception further). This delegation is also important in more complicated scenarios since the exception facing agent may not be in contact with all other agents in the protocol. In addition, trust is another important issue when considering multiple agents that enact a distributed protocol. It is more probable that an agent will respond to the agents it is committed to rather than other agents that it has no previous contact with. We aim to

investigate the application of trust strategies when considering such complicated scenarios.

ACKNOWLEDGEMENT

This research has been supported by Boğaziçi University Research Fund under grant BAP09A106P, The Scientific and Technological Research Council of Turkey by a CAREER Award under grant 105E073, and the Turkish State Planning Organization (DPT) under the TAM Project, number 2007K120610. We thank the anonymous referees for their comments on this paper.

REFERENCES

- [1] M. P. Singh, "An ontology for commitments in multiagent systems: Toward a unification of normative concepts," *Artificial Intelligence and Law*, vol. 7, pp. 97–113, 1999.
- [2] E. Giunchiglia, J. Lee, V. Lifschitz, N. McCain, and H. Turner, "Non-monotonic causal theories," *Artificial Intelligence*, vol. 153, no. 1-2, pp. 49–104, 2004.
- [3] A. K. Chopra and M. P. Singh, "Contextualizing commitment protocols," in *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous Agents and Multiagent Systems*. New York, NY, USA: ACM, 2006, pp. 1345–1352.
- [4] P. Yolum and M. P. Singh, "Flexible protocol specification and execution: applying event calculus planning using commitments," in *AAMAS '02: Proceedings of the first international joint conference on Autonomous Agents and Multiagent Systems*. New York, NY, USA: ACM, 2002, pp. 527–534.
- [5] N. Desai, A. K. Chopra, M. Arrott, B. Specht, and M. P. Singh, "Engineering foreign exchange processes via commitment protocols," in *International Conference on Services Computing (IEEE SCC)*, 2007, pp. 514–521.
- [6] A. U. Mallya and M. P. Singh, "Modeling exceptions via commitment protocols," in *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. New York, NY, USA: ACM, 2005, pp. 122–129.
- [7] M. Venkatraman and M. P. Singh, "Verifying compliance with commitment protocols," *Autonomous Agents and Multiagent Systems*, vol. 2, no. 3, pp. 217–236, 1999.
- [8] F. D. Jonge, N. Roos, and C. Witteveen, "Diagnosis of multi-agent plan execution," in *In Multiagent System Technologies: MATES 2006, LNCS 4196*. Springer, 2006, pp. 86–97.
- [9] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, and P. Torroni, "Verifiable agent interaction in abductive logic programming: The sciff framework," *ACM Transactions on Computational Logic*, vol. 9, no. 4, pp. 1–43, 2008.
- [10] A. Lomuscio, H. Qu, and M. Solanki, "Towards verifying compliance in agent-based web service compositions," in *Proceedings of 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2008, pp. 265–272.
- [11] F. Guerin and J. Pitt, "Agent communication frameworks and verification," in *AAMAS 2002 Workshop on Agent Communication Languages*, 2002.

A Methodology for Developing Self-Explaining Agents for Virtual Training

Maaïke Harbers^{1,2}, Karel van den Bosch¹ and John-Jules Meyer²

¹TNO Human Factors, P.O.Box 23, 3769 ZG Soesterberg, The Netherlands

²Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, The Netherlands

{maaike,jj}@cs.uu.nl, karel.vandenbosch@tno.nl

Abstract—Intelligent agents are used to generate the behavior of characters in virtual training systems. To increase trainees' insight in played training sessions, agents can be equipped with capabilities to explain the reasons for their actions. By using an agent programming language in which declarative aspects of an agent's reasoning process are explicitly represented, explanations revealing the underlying motivations for agents' actions can be obtained. In this paper, a methodology for developing self-explaining agents in virtual training systems is proposed, resulting in agents that can explain their actions in terms of beliefs and goals.

I. INTRODUCTION

Virtual training systems are often used to train people for complex, dynamic tasks in which fast decision making is required, e.g. the persons in command in crisis management, military missions or fire-fighting. During a training session, trainees interact with other virtual players, such as team-members, opponents, or colleagues from other domains. Using intelligent agents to generate the behavior of these virtual players lets trainees train at any place and time, reducing costs.

Typical mistakes of trainees include giving incomplete or unclear instructions, forgetting to monitor task execution, and failing to pick up new information and quickly adapt to it. Many of these errors involve situations in which a trainee makes false assumptions about other agents' knowledge or intentions. For example, a commanding fire-fighter who is in a fire truck to contact the dispatch center will not hear that one of his team members yelled that he saw a victim. His team members, however, might not have seen the commander in the truck and unjustly assume that he heard the message, leading to suboptimal behavior. Evidence for the origin of such mistakes can be found in literature; attributing incorrect knowledge and intentions to others is a well described phenomenon in cognitive sciences. For example, Nickerson gives an overview on literature about the tendency to ascribe one's own knowledge to others [11], and Keysar reports on limits in theory of mind use in practice, i.e. attributing incorrect mental states to others [6].

To improve trainees' performances, they should be made aware of their (possibly) false assumptions about others. Better understanding of past sessions should make trainees more alert and decrease the probability that they will make similar errors in a next incident. Therefore, we propose the use of self-explaining agents in virtual training, i.e. agents able to explain

the reasons for their actions. Humans usually explain and understand their own and others' behavior in terms of beliefs, desires and other mental contents [5]. Therefore, to provide useful explanations for humans, agents should explain their behavior in a similar terminology, e.g. by revealing the goals they held during a training session. Such explanations serve to increase the trainees' awareness of other agents' perspectives.

Current approaches of explanation in artificial intelligence do not provide explanations from an intentional perspective, that is, in terms of goals. Expert system explanations usually provide traces of the steps behind an diagnose or advice, and the justifications of those steps [17]. However, behavior of expert system is usually not understood in terms of intentions, in contrast to behavior of virtual characters. There are a few accounts of self-explaining agents in virtual training systems [8], [16], [3], but these do not provide information about the actual goals behind an agent's actions.

In order to explain agent behavior in terms of intentions, agents must be implemented in such a way that they act on the basis of intentions, and that their intentions are explicitly represented and thus available for the generation of explanations. In other words, behavior generation and explanation are connected; the reasoning steps that generate an action, are also used to explain that action. Making a connection between generation and explanation of behavior can be achieved by implementing self-explaining agents in a BDI-based agent programming language, because in those languages the declarative concepts needed for explanation are explicitly represented.

In this paper we introduce a methodology for developing self-explaining agents for virtual training systems. The different steps in the method are: determining the required scope of an agent's behavior (section II), constructing a task hierarchy of the agent (section III), implementing the agent in a BDI-based agent programming language (section IV), and then adding explanation facilities to the implementation (section V). Although explanation facilities are added to the agent only in the end, the programming language and methods in the previous steps were chosen in such a way that this would be possible. In the paper we discuss all four steps, but we most explicitly explain how a task hierarchy can be translated to a BDI-based agent program. Section VI discusses related work, and in section VII, we draw conclusions and give suggestions

for future research.

II. REQUIRED BEHAVIOR

To develop self-explaining agents for a virtual training system, knowledge about the required behavior and capacities of the agents is needed. Domain experts have knowledge about the tasks that belong to the jobs and roles of the agents, e.g. the tasks and responsibilities of a fire-fighter or an operator. Additionally, the training scenario(s) in which the agents will have to act give a lot of information about their required capacities. Training scenarios determine the scope of the situations in which the agents might arrive. For example, a firefighter might be responsible for the maintenance of tools, but tasks connected to this goal are not relevant in training scenarios about incidents, and thus do not have to be modeled.

One of the difficulties of writing a training scenario is to find a balance between freedom of the players (both agents and trainee) and continuation of a storyline, also called the *narrative paradox* [9]. On the one hand, trainees should be able to act as if it were a real situation and experience the consequences of their actions, e.g. believable reactions of the agents. On the other hand, because of inadequate acting of the trainee, the course of the scenario could change in such a way that situations in which specific learning goals can be trained do not occur. A possible solution of the narrative paradox in virtual training is to correct the trainee in a natural way if he deviates too much from the intended storyline, namely by directing the trainee with behavior of other players in the scenario. For instance, if a leading fire-fighter forgets to initiate smoke evacuation, initially nothing might happen, but eventually, a team member can ask for the smoke evacuation plan so that the trainee can practice to lead a smoke evacuation process. Thus, when determining the required capacities of an agent, tasks and actions involving the redirection of the trainee should be included.

A second aspect of importance for the scope of an agent's capacities concerns variation among different training scenarios. Most virtual training systems offer several training scenarios, to let trainees practice on different aspects of a mission. For instance, a fire-fighter might encounter incidents with or without victims, chemical substances or failing communication tools. Scenarios can be adjusted to the trainee's level of competence, e.g. scenarios are offered to the trainee with increasing difficulty. In conclusion, all possible scenario lines should be taken into account when determining the required capacities of an agent.

The required capacities of an agent in a scenario, including redirection actions and actions in different variations on the scenario, are input for the construction of a task hierarchy. A discussion on task hierarchies is given in the next section.

III. THE AGENT'S TASKS

Writing a training scenario lays down the possible *observable* actions of an agent, but actions are the result of *unobservable* processes leading to select those particular actions. Many processes could underlie the generation of an action,

but we believe the generation of behavior should be connected to behavior explanation. The deliberation steps that are taken to generate an action can also be best used to explain that action, and when these deliberation steps are understandable, the explanations should be as well. So while designing an agent with explanation capabilities, the unobservable internal processes should be meaningful.

In cognitive psychology, simple task analysis techniques restrict analyses to observable behavior, but cognitive task analysis also involves the knowledge, thought processes, and goal structures that underlie observable task performance. Hierarchical task analysis (HTA) is a well established technique [15], which smoothly connects observable behavior to internal cognitive processes by the systematical decomposition of goals or tasks. This feature makes it appropriate for developing self-explaining agents, who are supposed to explain the observable by the internal.

Many accounts of planning in artificial intelligence are based on hierarchical task representations, called hierarchical task networks (HTNs) [13]. In the strict sense, a HTN is the decomposition of an abstract task into more detailed subtasks; however, many accounts of HTN involve other features, e.g. information about which subtasks to select under given circumstances.

HTA and HTN planning both refer to a wide range of approaches, methods and techniques. In this paper, we leave it open *how* the task analysis should be made, but we do specify what should be the *result*. Namely, a task hierarchy represented in the task hierarchy representation language as introduced in the next section.

A. Task hierarchy representation language

A task hierarchy H in state S consists of a number of tasks which are related to each other by task-subtask relations. A task is defined as $T(\text{Type}, [(T1, C1), \dots, (Tn, Cn)])$, where Type denotes the type of task T , $[T1, \dots, Tn]$ are possible subtasks of task T , and $C1, \dots, Cn$ denote the conditions under which a subtask is adopted. Subtasks can in turn be decomposed into subtasks, etc. There are four types of tasks, namely *all*, *one*, *seq* and *prim*. Tasks of the type *all*, *one* or *seq* have subtasks, and a task's type denotes the relation to its subtasks. Tasks of the type *prim* are not decomposed, and can be achieved by executing a single action in the environment. In that case, the list with subtasks is empty: $T(\text{prim}, [])$.

Tasks can be adopted, which means that they are tried to achieve, and dropped, which means that they are either achieved or no longer tried to achieve. For all tasks but the top task it holds that a task can only be adopted when its main task is adopted, and the task is applicable, i.e. the conditions Ci connected to that task are consistent with state S . If there are no conditions connected to a task then $Ci = \text{true}$ and the task is always applicable. The three possible task-subtask relations *all*, *one* and *seq* each imply their own conditions to task adoption. A task-subtask relation of the type *all* means that all applicable subtasks are adopted. A task-subtask relation of the type *one* means that only one of the applicable subtasks

is adopted. The term *seq* refers to sequential and in such a task-subtask relation all subtasks are adopted, but only one by one in a specific order.

When a task is achieved it can be dropped. Task achievement depends either on state *S*, i.e. the conditions in the environment, or on achievement of a task's subtasks. An example of the first possibility is that the task to extinguish a fire is achieved when the fire is out. The second possibility occurs when the environment does not immediately give feedback on whether a task has been performed successfully. For instance, the task to report something can be achieved by sending an email, but one does not directly know if the email is read and understood. In such a cases task performance depends on task execution, that is, if a task is executed it is assumed to be achieved.

In our task hierarchy language, task achievement of primitive tasks always depends on task execution, and primitive tasks are thus always dropped after execution. Task achievement of non-primitive tasks either depends on conditions in state *S*, or on achievement of a task's subtasks. If the achievement of a task depends on the achievement of its subtasks, the relation with its subtasks defines the dependence relation. A task with a task-subtask relation of the type *all* is achieved when all applicable subtasks are achieved. A task with a task-subtask relation of the type *one* is achieved when exactly one of its subtasks is achieved. A task with a task-subtask relation of the type *seq* is achieved when all of its subtasks are achieved one by one, in the right order.

For each task holds that it either is achieved or not. There are several reasons for not allowing partially successful task executions. First, it is often hard to determine the measure of success of a task execution. Second, the task domains we aim at are of a procedural nature; if a task is not executed satisfactorily, alternative actions have to be taken, otherwise not. Third, the easiest way to represent partially succeeded tasks would be with a numerical approach, which makes it more difficult to provide explanations. The last two reasons together show that the domains we consider are appropriate for developing self-explaining agents.

B. An example: a firefighting agent

In this section we introduce an agent that could be one of the virtual characters in a scenario to train the head of a crisis management team. In such training, the trainee, who is playing the head, is confronted with a crisis he has to solve. A part of his tasks is to instruct and monitor the leaders of several teams, which are played by intelligent agents. The agent in this example is a firefighter, leading a firefighting team. The firefighting agent's tasks consist of receiving an attack plan from its head (the trainee) and pass corresponding instructions to its team, monitor the execution of the plan, and finally, report to the head that the incident has been solved.

Figure 1 shows the task hierarchy of the firefighting agent during the plan execution phase. Its two main occupations in this phase are checking the plan execution by its team, which involves dealing with a fire, victims and explosives,

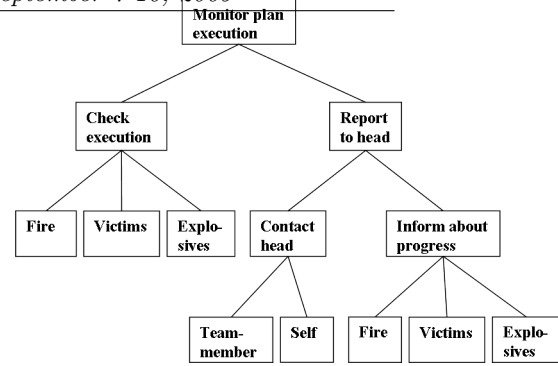


Fig. 1. Part of the firefighter's task hierarchy

and reporting its observations to the head. Usually, the firefighter reports several times to the head, and it can report on one or more aspects of the incident at the same time. A representation of the task hierarchy in figure 1 in the task hierarchy representation language just introduced looks as follows.

```

Monitor(all,[(Check(X),true),(Report,true)])
Check(X)(all,[(Check(Fire),true),(Check(Victims),true),
              (Check(Explosives),true)])
Check(Fire)(prim,[])
Check(Victims)(prim,[])
Check(Explosives)(prim,[])
Report(seq,[(ContactHead(Y),torep(X)),(Inform(X),incontact)])
ContactHead(Y)(one,[(ContactHead(Member),available(member)),
                    (ContactHead(Self),not available(member))]
ContactHead(Member)(prim,[])
ContactHead(Self)(prim,[])
Inform(X)(all,[(Inform(Fire),torep(Fire)),(Inform(Victims),
              torep(Victims)),(Inform(Explosives),torep(Explosives))])
Inform(Fire)(prim,[])
Inform(Victims)(prim,[])
Inform(Explosives)(prim,[])
    
```

In this representation the conditions *C* for task adoption and the task's relations *Type*, which are not shown in figure 1, are given. For example, *available(member)* means that there is a team member available, and *torep(X)* means that there is some information *X* that has not been reported to the head yet.

The *all* relation of the task Monitor denotes that the Check(X) and Report tasks are both adopted. To report, the firefighter has to contact the head and report about the different aspects of the incident. These two tasks have to be performed one by one and in this order, which is denoted by a *seq* relation. The fire-fighter agent can either contact the head itself, ContactHead(Self), or let a team member do it, ContactHead(Member). The *one* relation represents that executing one of these options should be sufficient to achieve contact with the head ContactHead(Y). Finally, the different aspects of the incident all have to be checked and reported, but the order in which they are checked and reported depends on how each of the aspects develops. This is denoted by an *all* relation.

IV. IMPLEMENTATION OF THE AGENT

The implementation of a self-explaining agent should fulfill four requirements. *First*, it should be possible to explicitly represent the agent's beliefs and goals. As stated in the introduction, the self-explaining agents should adopt the intentional stance to explain their behavior, i.e. explaining their actions in terms of goals. In section 3, we argued to connect the generation and explanation of behavior, e.g. if goal G makes an agent execute action A , G also explains why the agent executed A . Therefore, to let agents generate intentional explanations, they need to reason with beliefs and goals. The *second* requirement is that the operationalization of the agent's reasoning elements should be present in the implementation as well. Explanations should give insight into which elements relate to each other, how they relate to each other and how they interact. For instance, to achieve goal G it is necessary to achieve subgoal G_i . *Third*, a self-explaining agent should be able to introspect. An agent needs to have knowledge about its own states and processes in order to explain them. The *fourth* and last requirement is that a self-explaining agent needs to have memory. To explain its actions, an agent not only needs to have access to its states and processes at the time they occur, but also has to memorize them for possible future use.

Concerning the first requirement, a declarative agent programming language is needed. There are a number of agent programming languages based on the BDI (belief desire intention) model [12], which do allow for explicit representations of beliefs and goals. We have chosen to use 2APL [2] for our implementation as it connects declarative aspects like beliefs and goals to plans and actions, following from the interaction between beliefs and goals. Introspection is also possible in 2APL, the agent can perform checks on its belief and goal bases. A 2APL agent typically has no memory of its past states, processes and actions; these are only implicitly present in the interpreter. However, the agent's belief base can be considered as a memory, and a log about those aspects can be created in its belief base. For a more detailed overview of 2APL see [2].

A. Task hierarchies vs. BDI models

Our aim is to translate task hierarchy representations as discussed in section 3 to BDI-based agent programs. The most important similarity between task hierarchies and BDI models is that both reduce high-level entities into lower-level ones. A difference is that the first only contains tasks whereas the second makes a distinction between goals (desired world states) and plans (sequence of actions or sub-plans describing how a goal can be achieved). Table 1 shows the correspondence between the elements in a task hierarchy and a BDI-agent. An agent's main task must be implemented as a goal in order to generate plans and make the agent act, and primitive tasks can only be implemented as plans. However, the other tasks in the task hierarchy can be implemented as either plans or goals. In this section we discuss the advantages and disadvantages of representing tasks as goals or as plans.

Task hierarchy	BDI agent
State	Beliefs
Main task	Goal
Subtask	Goal or plan
Primitive task	Action (atomic plan)

TABLE I
TASK HIERARCHIES VS. BDI AGENTS

Beliefs and goals of an agent are related to each other in the sense that if an agent believes a certain fact, then the agent should not pursue that fact as a goal. In general, a goal remains in the agent's goal base until it is achieved, which is caused by sense actions leading to changes in the agent's belief base. In exceptional cases a goal can be explicitly dropped by the agent (as part of a plan). Plans, in contrast, are removed from an agent's plan base once they are executed. As a consequence, goals are more appropriate for the implementation of tasks which are achieved by an unknown number of actions (depending on the environment), e.g. monitoring plan execution.

Another difference between plans and goals concerns the way in which they are executed or achieved. The deliberation cycle of an agent states which step the agent should perform next, e.g. execute an action or apply a reasoning rule. In this cycle, the rules that are applicable to goals (PG-rules) are tried to be applied. But for plans, in contrast, plan by plan is considered which rules (PC-rules) apply to that plan. Thus, the order of goal execution depends on the order of the rules, whereas the order of plan execution depends on the order of the plans in the agent's plan base. As a programmer it is easier to exert control over the order of rules than over the order of plans in a plan base because an agent's rules remain the same, but its plans change during program execution.

For domains in which the number and order of tasks to be executed is fixed, it is easier to implement tasks as plans because the agent program ensures that plans are executed in the given order and dropped after execution automatically. In general, however, implementing tasks as goals allows for more flexibility because the number and order of actions to achieve a goal may vary. Therefore, we argue to implement all tasks in a hierarchy as goals, except for primitive tasks.

B. From task hierarchy to 2APL agent

In this section we discuss the translation of a task hierarchy representation as introduced in section III to 2APL code. In 2APL, an agent's beliefs are Prolog facts and rules, and the belief base of a 2APL agents is a Prolog program. Thus, from the beliefs x and $y :- x$, the belief y can be derived. To reason with goals, so called PG-rules are used, which are of the form $\text{Goal} \leftarrow \text{Belief} \mid \text{Plan}$. Informally this means that if the agent believes *Belief*, then to achieve the *Goal* it should execute *Plan*. To adopt a new goal (subtask), a *Plan* consists of the action *adopt(Subgoal)*, which means that the *Subgoal* is added to the agent's goal base. Dropping a goal is settled by beliefs in the agent's belief base. For each of the task types (all, one, seq, prim) we show the transition from task hierarchy representation to implementation. To ensure that the program 'walks through' the task hierarchy as desired,

we use the fact that the interpreter considers PG-rules from top to bottom. More specific rules are implemented above (and thus tried before) more general rules, so that the most specific rule as possible is applied.

a) All tasks: For tasks of the type *all*, all *applicable* subtasks are adopted. In the implementation, a PG-rule is added for each subtask, thus an *all*-task with *n* subtasks is implemented by *n* PG-rules. The task Monitor in our example has an *all* relation with its subtasks: Monitor(*all*, [(Check(*X*), true), (Report, true)]). The implementation looks as follows.

```
Monitor <- true | adopt(Check(X))
Monitor <- true | adopt(Report)
```

The first part of a rule is a check on the agent's goal base. Both PG-rules are only applied if Monitor is one of the agents goals. The second part of the PG-rule is a check on the agent's belief base. In this case, the guards of both rules are always true, i.e. the applicability of the subtasks is independent of the agent's beliefs. The body of the two rules states that the goals Check(*X*) and Report have to be adopted, respectively. If a subtask would only have to be adopted under certain circumstances, these conditions can be specified in the guard of the rule.

In 2APL, if a goal (a desired world state) is believed to be true, that goal is dropped. For some goals it holds that they are achieved if a certain situation in the environment is true. For example, the goal to extinguish a fire can be dropped when the agent believes that there is no fire.

```
ExtinguishFire :- not fire.
```

Other goals are achieved when its subtasks have been achieved. For those goals, to ensure that they are dropped when necessary, beliefs according to the following example should be in the agent's belief base.

```
Monitor :- Check(X), Report.
```

The code states that when both Check(*X*) and Report are finished successfully, the Monitor task can be dropped. If the goal Report would only be applicable under conditions *C*, the following rule would have to be added.

```
Report :- not C.
```

The rule ensures that if a subtask is not applicable (in the example when not *C*), that subtask does not have to be actively achieved in order to achieve its main task. Note that in some situations a task is achieved without executing any action, e.g. when there is no incident.

b) One tasks: For tasks of type *one* holds that only one of their subtasks is adopted. A *one*-task with *n* subtasks is implemented by *n* PG-rules. The task G(*one*,[(G1,C1),(G2,C2)]), for example, is implemented as follows.

```
G <- C1 | adopt(G1)
G <- C2 | adopt(G2)
```

C1 and C2 denote exclusive situations to ensure that only one sub-goal is adopted.

The goal *G* can be dropped either if a certain situation in the environment is true, or if one of the sub-goals is achieved. The last is implemented in the agent's belief base as follows.

```
G :- G1.
G :- G2.
```

Two separate beliefs are needed to express the dropping condition of goal *G* because *G* might be achieved by *G1* or by *G2*.

c) Seq tasks: For tasks of the type *seq*, all of their subtasks are adopted, but one by one and in a specific order. For example, the task Report(*seq*, [(ContactHead(*Y*), torep(*X*)), (Inform(*X*), incontact)]) is implemented as follows.

```
Report <- torep(X) | adopt(ContactHead(Y))
Report <- incontact | adopt(Inform(X))
```

The head of the rule contains the task for which subtasks need to be achieved. The guard of the rule contains conditions under which a rule can be adopted. As with tasks of the type *one*, the conditions specify unique circumstances here, so that only one subtask is executed at a time. Because the subtasks must be achieved in a specific order, the guards of the rules are beliefs related to goals, where the goal is the previous task in the sequence. For instance, the firefighter agent only starts to inform the head when it believes it is in contact. The belief torep(*X*) can be derived from the belief base if there is a priority that has been checked, but not yet been reported to the head, which is implemented as follows.

```
torep(X) :- Check(X), not Inform(X).
```

In general, a task with relation *seq* can be dropped if its last subtask is achieved.

```
Report :- Inform(X).
```

There is one exception. Namely, if the subtasks of a task of the type *seq* are primitive tasks, only one PG-rule is needed. For instance, a task with three subtasks who have to be executed one by one in a fixed order is implemented as follows.

```
Head <- Guard |
    { PrimTask1; PrimTask2; PrimTask3 }
```

As the order of the primitive subtasks is fixed and they can be executed immediately, it is not necessary to use different PG-rules. The different subtasks are added to the agent's plans base, and automatically executed in the right order.

d) Prim tasks: Primitive tasks are not divided into further subtasks and therefore implemented as plans. The following code shows an example of a task with the relation *one* to its primitive subtasks, namely ContactHead(*Y*)(*one*, [(ContactHead(Member), available(member)), (ContactHead(Self), not available(member))]).

```
ContactHead(Y) <- available(member) |
  ContactHead(Member)
ContactHead(Y) <- not available(member) |
  ContactHead(Self)
```

In this example the task `ContactHead(Y)` has two primitive subtasks. Instead of adopting a new goal, they can immediately be executed as actions in the environment. The implementation of primitive tasks with a parent task of the type *all* is similar to the example above.

As primitive tasks are implemented as plans, they are automatically removed from the agent's plan base once executed. If the goal for which they were executed is only dropped when certain conditions in the environment become true, they might be executed again. The other possibility is that their parent goal is dropped when (one of) its subtasks are (is) executed. In that case, the primitive task should involve an action in the environment, and add a belief to the agent's belief base which indicates that the action has been executed. In section V it will be explained how actual actions and belief update actions can be represented connected to each other, such that they are seen as one atomic action and their execution cannot be interrupted.

V. EXPLAINING ACTIONS

As stated before, self-explaining agents should be able to introspect and memorize; they need to have knowledge about their own past states and processes in order to explain them. In 2APL, the information required for explanations is present in the program and the interpreter, but not available to the agent for explanation at a later moment in time. One possibility is to investigate how the information present in the interpreter could be made accessible for explanation purposes, but in this paper we chose to focus on a solution that does not require adaptations to the programming language.

In order to reproduce past actions and motivations, an agent needs to store them at the time it has access to them, which is during task execution. The agent can store its present beliefs, goals and actions in a so-called explanation log in its belief base. Such a log can be created by connecting belief update actions to actual actions of the agent. For instance, when the agent adopts goal *G* at time *t*, it also logs that it adopted goal *G* at *t*.

```
Monitor <- true |
  [ adopt(Check(X));
    UpdateLog(Check(X), t) ]
```

The `[]` brackets in the code ensure that the execution of the two plans `adopt()` and `UpdateLog()` can not be interrupted by any other process; they are considered as one atomic plan. Actions that update an agent's explanation log can be connected any 'normal' action of the agent. Such update actions can of course be connected to all actions of the agent, but some updates may not be needed in an explanation. Therefore, the decision what to log and what not should depend on the information that is desired in an explanation.

In order to explain its behavior, an agent needs a log of past actions, but it also needs to have knowledge about its own task hierarchy. This knowledge is represented by beliefs of the type `Task(T,[T1,...,Tn])`, where *T1* to *Tn* are *T*'s subtasks. The agent has such a belief for each task it could possibly adopt. For example, our firefighting agent has the following beliefs about its task hierarchy in its belief base.

```
Task(Inform(X), [Inform(Fire),
  Inform(Victims), Inform(Explosives)]) .
Task(Report, [ContactHead(Y), Inform(X)]) .
Task(Monitor, [Check(X), Report]) .
```

With a combination of the beliefs containing the agent's complete task structure and beliefs that were logged during task execution, explanations of any action can be created. An extensive explanation is for example *I executed Inform(Victims) to achieve the goal Inform(X), which was executed to achieve the goal Report, which was executed to achieve the goal Monitor*. However, the complete trace of tasks responsible for one action might provide too much information; especially in bigger agent models it is crucial to make a selection of explaining elements. Such a selection consists of tasks either with a higher or a lower position in the hierarchy, yielding more abstract or specific explanations, respectively. An abstract explanation in this case would be: *I executed Inform(Victims) because I had the goal Monitor*. A specific explanation would be: *I executed Inform(Victims) because I had the goal Inform(X)*. More advanced explanation facilities could be interactive. For instance, the self-explaining agent starts with providing an abstract explanation, but if the trainee asks for extra information, more specific goals are provided.

VI. RELATED WORK

Already at an early stage in expert systems research it was recognized that advices or diagnoses from decision support systems should be accompanied by explanations [17], [4]. An often made distinction is that between rule trace and justification explanations [17]. Rule trace or *how* explanations show which rules or data a system uses to reach a conclusion. Justification or *why* explanations, in addition, provide the domain knowledge underlying these rules and decisions. Research shows that users of expert system often prefer *why* to *how* explanations. An important difference between expert systems and self-explaining agents is that the self-explaining agents are proactive, i.e. they have goals. Therefore, the goals because of which an agent executed an action also form the explanation of that action.

Explanations in virtual training systems are often provided by intelligent tutoring systems (ITSs), for an overview see [10]. However, ITSs mostly provide hints and eventually recipes of what is to be done. There are a few approaches of self-explaining agents in virtual training systems that provide explanations from the agents' perspectives. The Debrief system [8] explains actions by what *must have been* an agent's underlying beliefs. However, the explanations do not provide

the agent's actual beliefs or goals. The XAI system [16] provides explanations about agents' physical states, e.g. position and health, but not about their motivations. An improved version of the XAI system [3] aims to provide explanations of agents' motivations by making use of information made available by the simulation. However, simulations often do not provide information about agents' motivations, and if so, the explanations are not based on the actual motivations of the agents.

Our approach of planning has similarities with HTN-planning approaches. Currently, one of the most extensive accounts of general (HTN) planning is the GPGP approach [7]). GPGP (generalized partial global planning) is a framework for the coordination of small teams of agents. Our approach differs on the following aspects with the GPGP approach. First, whereas the GPGP approach is designed for coordination of a group of agents, our approach is designed for planning of a single agent. Second, GPGP explicitly defines non-local task structures: relations between two goals at any place in the goal tree. In our model these relations are not specified, but implicitly present in the adoption conditions of the goals. For instance, the condition to adopt a goal is the achievement of another goal. Finally, the GPGP approach involves partially successful tasks, whereas we only allow tasks to be successful or not. The reasons for this choice are explained in the last paragraph of section III-A.

Sardina et al also used the similarities between BDI systems and HTN planners, for an approach on planning [14]. They present formal semantics for a BDI agent programming language which incorporates HTN-style planning as a built-in feature. In contrast to their approach, we do not incorporate HTN-planning in a BDI-based agent program; instead, we give a mapping of the former to the latter.

Self-explaining agents need to have knowledge about their past mental states and actions. Such knowledge is called an episodic or autobiographic memory. Most research on agents with an episodic memory focuses on how the memory can improve an agent's performance. In our approach, agents do not use their episodic memory during task execution, but only afterwards, to explain their behavior. Nevertheless, our approach has similarities with the work of Brom et al [1] as they also use a hierarchical structure to model their agents, both in the agent program and the episodic memory.

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we have introduced a methodology for developing self-explaining agents in virtual training systems. The methodology involves: i) a determination of the agent's required behavior, ii) the construction of a task hierarchy, iii) an implementation of the self-explaining agent in a BDI-based agent programming language, and iv) the addition of explanation capabilities.

Single phenomena and processes can be explained in many different ways, but providing complete explanations is neither possible, nor desired [5]. By using a BDI-based approach, the scope of possible explanations is restricted; actions are

only explained in terms of beliefs and goals. Still, one action might have many underlying goals, and some selection on the information provided in an explanation might increase its effectiveness. For example, abstract explanations are given by just providing goals higher in the task hierarchy, and more specific explanations only consist of elements lower in the hierarchy. We are currently developing and implementing actual self-explaining agents for a specific virtual training system. When finished, we will be able to test the usefulness of the provided explanations, and investigate e.g. whether there is a general desired abstraction level of explanations.

ACKNOWLEDGMENTS

This research has been supported by the GATE project, funded by the Netherlands Organization for Scientific Research (NWO) and the Netherlands ICT Research and Innovation Authority (ICT Regie).

REFERENCES

- [1] C. Brom, K. Peskova, and J. Lukavsky. What does your actor remember? towards characters with a full episodic memory. In M. Cavazza and S. Donikian, editors, *Proc. of ICVS 2007*, pages 89–101. Springer-Verlag Berlin Heidelberg, 2007.
- [2] M. Dastani. 2APL: a practical agent programming language. *Autonomous Agents and Multi-agent Systems*, 16(3):214–248, 2008.
- [3] D. Gomboc, S. Solomon, M. G. Core, H. C. Lane, and M. van Lent. Design recommendations to support automated explanation and tutoring. In *Proc. of the 14th Conf. on Behavior Representation in Modeling and Simulation*. Universal City, CA., 2005.
- [4] S. Gregor and I. Benbasat. Explanation from intelligent systems: theoretical foundations and implications for practice. *MIS Quarterly*, 23(4):497–530, 1999.
- [5] F. Keil. Explanation and understanding. *Annual Reviews Psychology*, 57:227–254, 2006.
- [6] B. Keysar, S. Lin, and D. Barr. Limits on theory of mind use in adults. *Cognition*, 89:25–41, 2003.
- [7] V. Lesser, K. Decker, N. Carver, A. Garvey, D. Neiman, M. Nandras Prasad, and T. Wagner. Evolution of the GPGP/TAEMS domain-independent coordination framework. *Autonomous agents and multi-agent systems*, 9:87–143, 2004.
- [8] W. Lewis Johnson. Agents that learn to explain themselves. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence*, pages 1257–1263, 1994.
- [9] S. Louchart and R. Aylett. Managing a non-linear scenario - a narrative evolution. In *Virtual Storytelling*, pages 148–157. Springer Berlin, 2005.
- [10] T. Murray. Authoring intelligent tutoring systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education*, 10:98–129, 1999.
- [11] S. Nickerson. How we know -and sometimes misjudge- what others know: Imputing one's own knowledge to others. *Psychological Bulletin*, 125(6):737–759, 1999.
- [12] A. Rao and M. Georgeff. Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proc. of the 2nd Internat. Conf. on Principles of Knowledge Representation and Reasoning*, pages 473–484, San Mateo, CA, USA, 1991. Morgan Kaufmann publishers Inc.
- [13] S. Russell and P. Norvig. *Artificial Intelligence A Modern Approach*. Pearson Education, Inc., New Jersey, USA, second edition, 2003.
- [14] S. Sardina, L. De Silva, and L. Padgham. Hierarchical planning in BDI agent programming languages: A formal approach. In *Proc. of AAMAS 2006*. ACM Press, 2006.
- [15] J. Schraagen, S. Chipman, and V. Shalin, editors. *Cognitive Task Analysis*. Lawrence Erlbaum Associates, Mahway, New Jersey, 2000.
- [16] M. Van Lent, W. Fisher, and M. Mancuso. An explainable artificial intelligence system for small-unit tactical behavior. In *Proc. of IAAA 2004*, Menlo Park, CA, 2004. AAAI Press.
- [17] R. Ye and P. Johnson. The impact of explanation facilities on user acceptance of expert systems advice. *Mis Quarterly*, 19(2):157–172, 1995.

An Integrated Semantics of Social Commitments and Associated Operations

M. El-Menshawy

Depart. of Electrical and Computer Engineering
Concordia University, Montreal, Canada
m_elme@encs.concordia.ca

J. Bentahar, R. Dssouli

Concordia Institute for Inf. Sys. Engineering
Concordia University, Montreal, Canada
{bentahar, dssouli}@ciise.concordia.ca

Abstract—In this paper, we develop a unified semantic model for social commitments and associated operations. We propose a logical model based on CTL^* with modalities of commitments and associated operations that represent the dynamic behavior of agents. Our semantics differs from the previous proposals in which the operations used to manipulate commitments (e.g. creation, fulfillment, violation, withdrawn, etc.) have always been defined as axioms or constraints on top of the commitment semantics. The advantage of this logical model is to gather the direct semantics of these operations and the semantics of social commitments (propositional and conditional) within the same framework. Furthermore, this paper proposes a new definition of assignment and delegation operations by looking at the content of the assigned and delegated commitment that could be different from the content of the original commitment in terms of deadline. Finally, to stress the soundness of the model, we prove that the proposed semantics satisfies some properties that are desirable when modeling commitment-based multiagent systems.

I. INTRODUCTION

The importance of defining suitable and formal semantics of social commitments has been broadly recognized for multi-agent systems. Particularly, social commitments have been used for agent communication, coordination and artificial institutions [22]. In communication protocols, commitments can capture a high level meaning of interaction as opposed to low-level operational representation [24], [14], [9]. However, the meanings of messages exchanged between interacting agents can also be interpreted directly into social commitments in an operational semantics style [11], [6].

Some interesting semantic frameworks for social commitments have already been proposed using different approaches such as branching time logics (CTL^* and CTL^\pm) [2], [3], [13], [19], [25]. Recently, a model-theoretic semantics of social and dialogical commitments based on linear-time logic (LTL) has been introduced in [20] and the proposed postulates are reproduced in [7] to define semantics of commitment operations in distributed systems.

In general, two categories of semantic frameworks for social commitments can be distinguished. In the former category, commitment operations are formalized based on Singh's presentation [18] as axioms or constraints on top of commitment semantics [6], [9], [13], [14], [24]. These axioms are represented either as reasoning rules, updating rules or enforcing rules to evolve the truth of commitments' states. However, the

real meanings of commitment operations themselves (e.g. *Create*, *Fulfill*, etc.) are not captured. In the later category, social commitments are formalized using object-oriented paradigm to advance the idea of commitments as data structure [11]. Thus, the main objective of defining clear, practical, and verifiable semantics of commitments and associated operations in the same framework is yet to be reached.

The objective of this paper aims to propose a new semantics not only for social commitments, but also for all the operations used to manipulate commitments. For verification issues and development methodologies of agent-oriented software engineering, the semantics of these operations should not be only captured by some enforced rules like in [13], but also integrated in the same framework [15]. In essence, this work is a continuation of our two previous publications [3], [15]. In the former one [3], we have developed a framework unifying commitments, actions on commitments and arguments that agents use to support their actions. In the second one [15], we have proposed a new logical semantics of social commitments and associated two-party operations based on Branching Space-Time (BST) logic. BST-logic provides this semantics with agent life cycle, space-like dimension and causal relation between interacting agents in the same (physical or virtual) space. Specifically, here we refine the semantics of some operations (e.g., *Create*, *Withdraw*, *Fulfill*) to overcome the *state explosion* problem that arises in [3], complete the life cycle of commitment operations introduced in [15], and define a new semantics of multi-party operations, such as *Delegate* and *Assign* using an extension of CTL^* .

The primary contribution of this paper is the unified logical model for commitments and associated operations. In fact, the paper introduces a new semantics of withdrawal, fulfilment, violation and release operations using the notions of accessible and non-accessible paths. New definitions of assignment and delegation operations are also proposed by taking into account the fact that the assigned and delegated commitment's deadline could be different from the deadline of the original commitment. Some desirable properties such as "the same social commitment (with the same identifier) cannot be created twice" or "if it is fulfilled, the commitment cannot be fulfilled again or withdrawn in the future" are captured, which makes the model sound. The proposed logical model uses a Kripke-like computational structure where accessibil-

ity relations for commitment modalities are defined using a computational interpretation, which makes our semantics computationally grounded [23] (this idea will be detailed later). This computational interpretation is suitable for formal verification using model checking to verify interacting agent-based systems against given properties [4], but model checking algorithm is out of scope of this paper. Furthermore, the logical model presented here is expressive because the content of commitments are CTL*-like path formulae [10] and not state formulae and their semantics is expressed in terms of accessible paths and not in terms of deadlines.

The remainder of this paper is organized as follows. Section II describes the notion of social commitment and its formal notation extended from [15]. Given this context, Section III presents the syntax and semantics of the main elements of our logical model. Subsequently, Section IV proves some logical properties based on the defined semantics. Finally, the paper ends in Section V with a discussion of related work.

II. SOCIAL COMMITMENTS

A. Formal Notation

A commitment is an engagement made by one agent, the *debtor*, and directed towards another agent, the *creditor*, so that some fact is true. The debtor must respect and behave in accordance with his commitments. These commitments are contextual, manipulable and possibly conditional [18]. Furthermore, commitments are social and observable by all the participants. Consequently, social commitments (SC) are different from the agent's private mental states like beliefs, desires and intentions. Several approaches assume that agents will respect their commitments. However, this assumption is not always guaranteed in real-life scenarios (e.g in e-business) since violation can occur if agents are malicious, deceptive, malfunctioning or not reliable. Thus, it is natural to introduce violation operation of social commitments along with their satisfaction. We can also use a legal context of commitments to define rules that impose penalties on the debtors that violate their commitments. Below, we distinguish between two types of social commitments: propositional and conditional.

Definition 1: *Propositional social commitments are related to the state of the world and denoted by $SC^p(id, Ag_1, Ag_2, \phi)$ where id is the commitment's identifier, Ag_1 is the debtor, Ag_2 is the creditor and ϕ is a well-formed formula (expressed in some logics) representing the commitment content.*

The basic idea is that Ag_1 is committed towards Ag_2 that the propositional formula ϕ is true. We suppose that the identifier id is unique so that there is at most one commitment with a particular identifier. In several situations, agents can only commit when some conditions are satisfied. Conditional commitments are introduced to capture this issue.

Definition 2: *Conditional social commitments are denoted by $SC^c(id, Ag_1, Ag_2, \tau, \phi)$ where $id, Ag_1, Ag_2,$ and ϕ have*

the same meanings as in Def.1 and τ is a well-formed formula representing the condition.

B. Social Commitment Life Cycle

Having explained the formal definitions of commitments, in this section we present their life cycle to specify the relationship between commitments' states. Figure 1 describes this life cycle using UML state diagram. The life cycle proceeds as follows:

- The commitment could be *conditional* or *unconditional* (i.e. propositional). This is represented by the selection operator. The first operation an agent can perform on a commitment is creation. When created, a conditional commitment can move to the state of *created unconditional commitment* if the condition is true. Otherwise, the conditional commitment moves to the final state.
- When the unconditional commitment is created, then it may move to one of the following states: *fulfilled*, *violated*, *withdrawn*, *released*, *delegated* or *assigned*.
- The commitment can be *withdrawn* if the debtor decides to cancel it. Only the debtor is able to perform this action without any intervention from the creditor.
- The commitment is *fulfilled* if its content is satisfied by the debtor.
- The social commitment is *violated* if its content is violated by the debtor.
- The social commitment can be *released* by the creditor so that the debtor is no longer obliged to carry out his commitment.

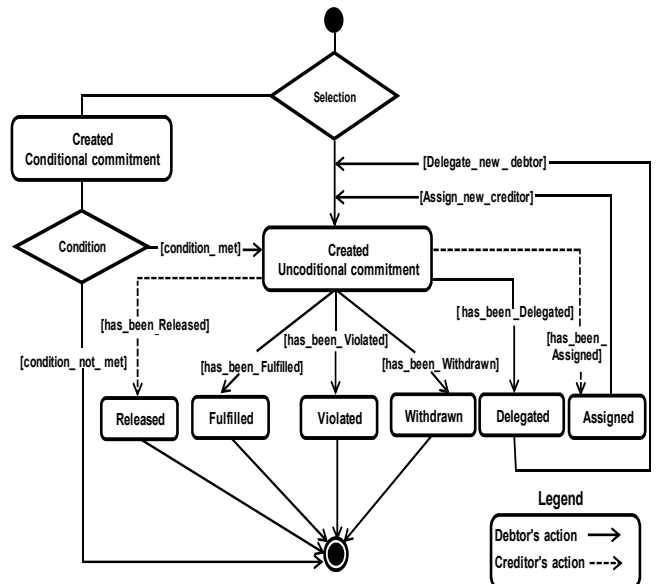


Fig. 1. Life cycle of social commitments

- The social commitment can be *assigned* by the creditor, which results in releasing this creditor from the commitment and having a new unconditional commitment with a new creditor.

- The social commitment can be *delegated* by the debtor, which results in withdrawing this debtor from the commitment and delegating his role to another debtor within a new commitment.

III. LOGICAL MODEL OF SOCIAL COMMITMENTS

This section introduces the syntax and semantics of the different elements of our formal language \mathcal{L} . This propositional language uses extended Computation Tree Logic (CTL*)[10] with past operators and two additional modalities SC^p for propositional and SC^c for conditional commitments, and Act for actions applied to commitments. We refer to the resulted branching time logic as CTL^{*sc} . The time in our model is discrete and branching in future to represent all choices that agents have when they participate in conversations and linear in the past. On the other hand, the dynamic behavior of agents is captured by actions these agents perform on different commitments during conversations.

A. Syntax

Let Φ_p be a set of atomic propositions and \mathcal{ID} be a set of identifiers. AGT is a set of agent names and ACT is a set of actions used to manipulate commitments (e.g. *Create*, *Fulfill*, etc.). L and Act are nonterminals corresponding to \mathcal{L} and ACT , respectively. Furthermore, we use the following conventions: id, id_0, id_1 , etc. are unique commitment identifiers in \mathcal{ID} , Ag_1, Ag_2, Ag_3 , etc. are agent names in AGT , p, q , etc. are atomic propositions in Φ_p and ϕ, ψ , etc. are formulae in \mathcal{L} . Table I gives the formal syntax of \mathcal{L} expressed in Backus-Naur Form (BNF) grammar where “ \rightarrow ” and “|” are meta-symbols of this grammar.

L1. $L \rightarrow C \mid Act \mid p \mid \neg L \mid L \vee L \mid X^+L \mid X^-L \mid L U^+L \mid$ $L U^-L \mid AL \mid EL$
L2. $C \rightarrow SC^p(id, Ag_1, Ag_2, L) \mid SC^c(id, Ag_1, Ag_2, L, L)$
L3. $Act \rightarrow Create(Ag, C) \mid Fulfill(Ag, C) \mid Violate(Ag, C)$ $\mid Withdraw(Ag, C) \mid Release(Ag, C)$ $\mid Assign(Ag_1, Ag_2, C) \mid Delegate(Ag_1, Ag_2, C)$

TABLE I
THE SYNTAX OF CTL^{*sc} LOGIC

The intuitive meanings of the most constructs are straightforward (from CTL* with next (X^+), previous (X^-), until (U^+), and since (U^-) operators). A and E are the universal and existential path-quantifiers over the set of all paths starting from the current moment. $A\phi$ (resp. $E\phi$) means that ϕ holds along all (some) paths starting at the current moment. Furthermore, there are some useful abbreviations based on temporal operators (X^+, X^-, U^+, U^-): (sometimes in the future) $F^+\phi \triangleq true U^+\phi$; (sometimes in the past) $F^-\phi \triangleq true U^-\phi$; (globally in the future) $G^+\phi \triangleq \neg F^+\neg\phi$ and (globally in the past) $G^-\phi \triangleq \neg F^-\neg\phi$. We also introduce $\mathcal{L}^- \subset \mathcal{L}$ as the subset of all formulae without temporal operators.

B. Semantics of CTL^{*sc}

1) The formal model:

Our model M for \mathcal{L} is based on a Kripke-like structure with seven-tuple, $M = \langle \mathbb{S}, T, R, \mathbb{V}, \mathbb{R}_{scp}, \mathbb{R}_{scc}, \mathbb{F} \rangle$, where:

- $\mathbb{S} = \{s_0, s_1, s_2, \dots\}$ is a set of states.
- $T : \mathbb{S} \rightarrow \mathcal{TP}$ is a function assigning to any state the corresponding time stamp where \mathcal{TP} is a set of time points.
- $R \subseteq \mathbb{S} \times \mathbb{S}$ is a total transition relation, that is, $\forall s_i \in \mathbb{S}, \exists s_j \in \mathbb{S} : (s_i, s_j) \in R$, indicating branching time. If there exists a transition $(s_i, s_j) \in R$, then we have $T(s_i) < T(s_j)$. A path P is an infinite sequence of states $P = \langle s_0, s_1, s_2, \dots \rangle$ where $\forall i \in \mathbb{N}, (s_i, s_{i+1}) \in R$. We denote the set of all paths by σ and the set of all paths starting at s_i by σ^{s_i} .
- $\mathbb{V} : \Phi_p \rightarrow 2^{\mathbb{S}}$ is a valuation function that assigns to each atomic proposition a set of states where the proposition is true.
- $\mathbb{R}_{scp} : \mathbb{S} \times AGT \times AGT \rightarrow 2^\sigma$ is a function producing an accessibility modal relation for propositional commitments.
- $\mathbb{R}_{scc} : \mathbb{S} \times AGT \times AGT \rightarrow 2^\sigma$ is a function producing an accessibility modal relation for conditional commitments.
- $\mathbb{F} : \mathcal{L} \rightarrow \mathcal{L}^-$ is a function associating to each formula in \mathcal{L} a corresponding formula in \mathcal{L}^- .

The function \mathbb{R}_{scp} associates to a state s_i the set of paths starting at s_i along which an agent commits towards another agent. These paths are conceived as merely “possible”, and as paths where the commitments’ contents made in s_i are true. The computational interpretation of this accessibility relation is as follows: the paths over the model M are seen as computations, and the accessible paths from a state s_i are the computations satisfying (i.e. computing) the formulae representing the contents of the commitments made at that state by a given agent towards another given agent. For example, if we have: $P' \in \mathbb{R}_{scp}(s_i, Ag_1, Ag_2)$, then the commitments that are made in the state s_i by Ag_1 towards Ag_2 are satisfied along the path $P' \in \sigma^{s_i}$.

\mathbb{R}_{scc} is similar to \mathbb{R}_{scp} and it gives us the paths along which the resulting unconditional commitment is satisfied if the underlying condition is true. Because it is possible to decide if a path satisfies a formula (see the semantics in this section), the model presented here is computationally grounded [23]. In fact, the accessible relations map commitment content formulae into a set of paths that simulate the behavior of interacting agents. The logic of propositional and conditional commitments is KD4 modal logic and the accessibility modal relations \mathbb{R}_{scp} and \mathbb{R}_{scc} are serials [3]. The function \mathbb{F} is used to remove the temporal operators from a formula in \mathcal{L} . For example: $\mathbb{F}(X^+X^+p) = p$ and $\mathbb{F}(SC^p(id, Ag_1, Ag_2, X^+q)) = SC^p(id, Ag_1, Ag_2, q)$.

2) Semantics of social commitments:

Having explained our formal model, in this section we define

the semantics of the elements of \mathcal{L} relative to a model M , state s_i , and path P . The notation $\langle s_i, P \rangle$ refers to the path P starting at s_i meaning that $P \in \sigma^{s_i}$ where $P = \langle s_i, s_{i+1}, s_{i+2}, \dots \rangle$. If P is a path starting at a given state s_i , then *prefix* of P starting at a state s_j ($T(s_j) < T(s_i)$) is a path denoted by $P \downarrow s_j$ and *suffix* of P starting at a state s_k ($T(s_i) < T(s_k)$) is a path denoted by $P \uparrow s_k$. Because the past is linear, s_j is simply a state in the unique past of s_i such that P is a part of $P \downarrow s_j$. s_k is in the future of s_i over the path P such that $P \uparrow s_k$ is part of P . If s_i is a state, then we assume that s_{i-1} is the previous state in the linear past ($(s_{i-1}, s_i) \in R$) and s_{i+1} is the next state on a given path ($(s_i, s_{i+1}) \in R$).

In the metalanguage, we use the following symbols: means “and”, \Leftrightarrow means “is equivalent to” and \Rightarrow means “implies that”. The logical equivalence is denoted \equiv . As in CTL^* , we have two types of formulae: state formulae evaluated over states and path formulae evaluated over paths [10]. $M, \langle s_i \rangle \models \phi$ means “the model M satisfies the state formula ϕ at s_i ”. $M, \langle s_i, P \rangle \models \phi$ means “the model M satisfies the path formula ϕ along the path P starting at s_i ”. A state formula ϕ is satisfiable iff there are some M and s_i such that $M, \langle s_i \rangle \models \phi$. A path formula ϕ is satisfiable iff there are some M, P , and s_i such that $M, \langle s_i, P \rangle \models \phi$. A state formula is valid when it is satisfied in all models M , in all states s_i in M . A path formula is valid when it is satisfied in all models M , in all paths P in M , in all states s_i . The formal semantics of CTL^* and SC^p, SC^c modalities of our model is illustrated in Table II.

-
- M1.** $M, \langle s_i \rangle \models p$ iff $s_i \in \mathbb{V}(p)$ where $p \in \Phi_p$
M2. $M, \langle s_i \rangle \models \neg \phi$ iff $M, \langle s_i \rangle \not\models \phi$
M3. $M, \langle s_i \rangle \models \phi \vee \psi$ iff $M, \langle s_i \rangle \models \phi$ or $M, \langle s_i \rangle \models \psi$
M4. $M, \langle s_i \rangle \models A\phi$ iff $\forall P \in \sigma^{s_i} : M, \langle s_i, P \rangle \models \phi$
M5. $M, \langle s_i \rangle \models E\phi$ iff $\exists P \in \sigma^{s_i} : M, \langle s_i, P \rangle \models \phi$
M6. $M, \langle s_i \rangle \models SC^p(id, Ag_1, Ag_2, \phi)$ iff
 $\forall P \in \mathbb{R}_{scp}(s_i, Ag_1, Ag_2) M, \langle s_i, P \rangle \models \phi$
M7. $M, \langle s_i \rangle \models SC^c(id, Ag_1, Ag_2, \tau, \phi)$ iff
 $\forall P \in \mathbb{R}_{sc}(s_i, Ag_1, Ag_2) M, \langle s_i, P \rangle \models \tau$
 $\Rightarrow M, \langle s_i, P \rangle \models SC^p(id, Ag_1, Ag_2, \phi)$
M8. $M, \langle s_i, P \rangle \models \phi$ iff $M, \langle s_i \rangle \models \phi$
M9. $M, \langle s_i, P \rangle \models \neg \phi$ iff $M, \langle s_i, P \rangle \not\models \phi$
M10. $M, \langle s_i, P \rangle \models \phi \vee \psi$ iff $M, \langle s_i, P \rangle \models \phi$ or $M, \langle s_i, P \rangle \models \psi$
M11. $M, \langle s_i, P \rangle \models X^+\phi$ iff $M, \langle s_{i+1}, P \uparrow s_{i+1} \rangle \models \phi$
M12. $M, \langle s_i, P \rangle \models \phi U^+ \psi$ iff $\exists j \geq i : M, \langle s_j, P \uparrow s_j \rangle \models \psi$ &
 $\forall i \leq k < j M, \langle s_k, P \uparrow s_k \rangle \models \phi$
M13. $M, \langle s_i, P \rangle \models X^-\phi$ iff $M, \langle s_{i-1}, P \downarrow s_{i-1} \rangle \models \phi$
M14. $M, \langle s_i, P \rangle \models \phi U^- \psi$ iff $\exists j \leq i : M, \langle s_j, P \downarrow s_j \rangle \models \psi$ &
 $\forall j < k \leq i M, \langle s_k, P \downarrow s_k \rangle \models \phi$
-

TABLE II
SEMANTICS OF CTL^* AND SC^p, SC^c MODALITIES

The semantics of state formulae is given from $M1$ to $M7$ and that of path formulae is given from $M8$ to $M14$. For space limit reasons, we only explain the semantics of formulae that are not in CTL^* . $M6$ gives the semantics of propositional commitment operator, where the state formula is satisfied in

the model M at s_i iff the content ϕ is true in all accessible paths P starting at s_i . Similarly, $M7$ gives the semantics of conditional commitment operator, where the state formula is satisfied in the model M at s_i iff in all accessible paths P if the condition τ is true, then the underlying unconditional commitment is also true. The semantics of past operators X^- and U^- is given by considering the linear past of the current state s_i as prefix of the path P .

3) Semantics of actions on social commitments:

Having defined the semantics of commitments, below we define the semantics of operations used to manipulate those commitments and support flexibility. These operations are of two categorizes: two-party operations: *Create, Withdraw, Fulfill, Violate* and *Release*, and three-party operations: *Assign* and *Delegate* because *Assign* and *Delegate* need a third agent to which the new commitment is assigned or delegated. The context and detailed exposition of these operations are given in [13], [14], [18]. To simplify the notations used in the semantics, we suppose that these actions are momentary and do not need time to be performed. Technically, this can be represented by allowing actions to be performed on states or by supposing that transitions labeled by these actions are connecting two states s_i and s_j having the same time stamp ($T(s_j) = T(s_i)$). The first possibility is adopted in this paper. Although actions are momentary, action formulae are evaluated over paths. This is compatible with the philosophical interpretation of actions, according to which, by performing an action the agent selects a path or history among the available paths or histories at the moment of performing the action.

Creation action: the semantics of creation action of a propositional commitment (see Table III) is satisfied in the model M at state s_i along path P iff (i) the commitment is established in s_i (as a result of performing the momentary creation action); and (ii) the created commitment was not established in the past.

-
- M15.** $M, \langle s_i, P \rangle \models Create(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))$ iff
 (i) $M, \langle s_i, P \rangle \models SC^p(id, Ag_1, Ag_2, \phi)$ &
 (ii) $\forall j < i, M, \langle s_j \rangle \models \neg SC^p(id, Ag_1, Ag_2, \phi)$
-

TABLE III
SEMANTICS OF CREATION ACTION RELATIVE TO SC^p

The semantics of creation action of a conditional commitment (see Table IV) is defined in the same way.

-
- M16.** $M, \langle s_i, P \rangle \models Create(Ag_1, SC^c(id, Ag_1, Ag_2, \tau, \phi))$ iff
 (i) $M, \langle s_i, P \rangle \models SC^c(id, Ag_1, Ag_2, \tau, \phi)$ &
 (ii) $\forall j < i, M, \langle s_j \rangle \models \neg SC^c(id, Ag_1, Ag_2, \tau, \phi)$
-

TABLE IV
SEMANTICS OF CREATION ACTION RELATIVE TO SC^c

Example 1: Let us consider a simple and modified case of NetBill protocol to illustrate the semantics of different action

formulae. A customer (*Cus*) requests a quote for some goods (rfq), followed by the merchant (*Mer*) sending the quote as an offer. If the customer pays for goods, then the merchant will deliver the goods, withdraw (within a specified time), or not deliver. The customer can also release after receiving the quote (see Fig.2).

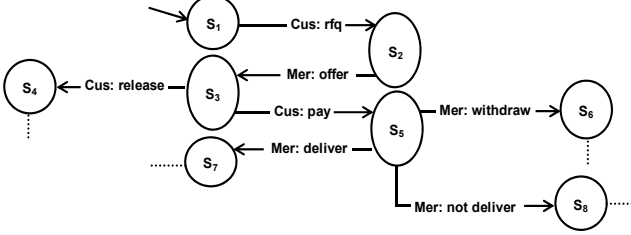


Fig. 2. Representation of NetBill

The offer message at state s_2 means that *Mer* creates a conditional commitment $Create(Mer, SC^c(id, Mer, Cus, pay, delivergoods))$ meaning that if the payment is received, then *Mer* commits to deliver the goods to *Cus*. $\langle s_2, s_3, s_4, \dots \rangle$, $\langle s_2, s_3, s_5, s_6, \dots \rangle$ and $\langle s_2, s_3, s_5, s_8, \dots \rangle$ are not accessible paths for this commitment (i.e. are not in $\mathbb{R}_{sc}(s_2, Mer, Cus)$). However, $\langle s_2, s_3, s_5, s_7, \dots \rangle$ is an accessible path (i.e. is in $\mathbb{R}_{sc}(s_2, Mer, Cus)$). As the condition is true through $\langle s_2, s_3, s_5, s_7, \dots \rangle$ (the customer pays), the conditional commitment becomes unconditional: $SC^p(id, Mer, Cus, delivergoods)$ along the same accessible path. Before creating this unconditional commitment, *Mer* checks that it has not been created before, as there is no reason to create the same commitment twice.

Withdrawal action: the semantics of withdrawal action of a propositional commitment (see Table V) is satisfied in the model M at s_i along path P iff (i) the commitment was created in the past at s_j through the prefix $P \downarrow s_j$; (ii) this prefix is not one of the accessible paths via \mathbb{R}_{sc} ; and (iii) at the current state s_i , there is still a possibility of satisfying the commitment since there is a path P' whose the prefix $P' \downarrow s_j$ is still accessible using \mathbb{R}_{sc} . Note that the first argument of \mathbb{R}_{sc} is s_j where the commitment has been created. This is because the accessible paths start from the state where the commitment is created. Intuitively, when a commitment is withdrawn along a path, the prefix of this path from the creation state does not correspond to an accessible path (condition ii).

M17. $M, \langle s_i, P \rangle \models Withdraw(Ag_1, SC^p(id_1, Ag_1, Ag_2, \phi))$ iff
 (i) $\exists j < i : M, \langle s_j, P \downarrow s_j \rangle \models Create(Ag_1, SC^p(id_1, Ag_1, Ag_2, \phi))$ &
 (ii) $P \downarrow s_j \notin \mathbb{R}_{sc}(s_j, Ag_1, Ag_2)$ &
 (iii) $\exists P' \in \sigma^{s_i} : P' \downarrow s_j \in \mathbb{R}_{sc}(s_j, Ag_1, Ag_2)$

TABLE V
SEMANTICS OF WITHDRAWAL ACTION

Furthermore, a commitment can be withdrawn when its satisfaction is still possible (condition iii), which is captured

by the existence, starting at the current moment, of an accessible path the agent can choose (see Fig.3). In other words, the agent Ag_1 has another choice at the current state, which is continuing in the direction of satisfying its commitment. We also note that withdrawing a commitment does not mean its content is false. For instance it can be accidentally true even if the current path is not amongst the accessible ones.

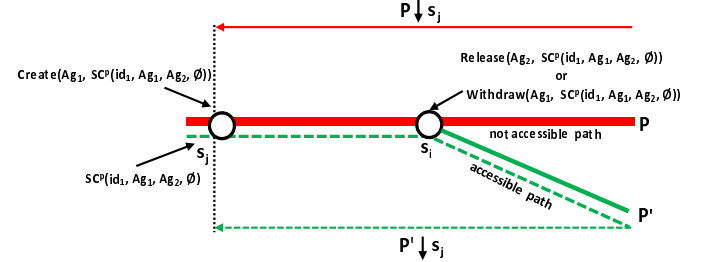


Fig. 3. Withdraw and Release actions at the state s_i along the path P

Example 2: The merchant *Mer*, before delivering the goods to *Cus*, can withdraw the offer. Thus, there is no accessible path for the commitment between *Mer* and *Cus* at s_6 . At the same time, *Mer* still has a possibility to satisfy its offer at state s_5 through the accessible paths $\langle s_2, s_3, s_5, s_7, \dots \rangle$.

Fulfillment action: the semantics of fulfillment action (see Table VI) is defined in the same way as withdrawal. In (ii), the prefix $P \downarrow s_j$ of the current path P (along which the commitment has been created) is accessible using \mathbb{R}_{sc} ; and in (iii), at the current state s_i , there is still a possible choice of not satisfying the commitment since a non-accessible path $P' \downarrow s_j$ exists. We notice that being accessible means that the content ϕ is true along $P \downarrow s_j$. As for withdrawal, fulfillment action makes sense only when a non-fulfilment action is still possible.

M18. $M, \langle s_i, P \rangle \models Fulfill(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))$ iff
 (i) $\exists j < i : M, \langle s_j, P \downarrow s_j \rangle \models Create(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))$ &
 (ii) $P \downarrow s_j \in \mathbb{R}_{sc}(s_j, Ag_1, Ag_2)$ &
 (iii) $\exists P' \in \sigma^{s_i} : P' \downarrow s_j \notin \mathbb{R}_{sc}(s_j, Ag_1, Ag_2)$

TABLE VI
SEMANTICS OF FULFILLMENT ACTION

Example 3: When the customer *Cus* pays for the goods and the merchant *Mer* delivers the goods (within a specified time), the merchant satisfies his commitment through the accessible path $\langle s_2, s_3, s_5, s_7, \dots \rangle$. At the moment of satisfying the commitment, the merchant has still a possibility of not satisfying it through the non-accessible path $\langle s_2, s_3, s_5, s_6, \dots \rangle$.

Violation action: the semantics of violation action (see Table VII) is almost similar to the semantics of withdrawal. The main difference is related to the truth of the commitment's

content, which is false in the case of violation (condition *ii*). The fact that ϕ is false implies that $P \downarrow s_j$ is not accessible, but the reverse is not always true as explained above. Here again, violation makes sense when a choice of satisfying the commitment is still possible at the current state (condition *iii*).

-
- M19.** $M, \langle s_i, P \rangle \models \text{Violate}(Ag_1, SC^P(id, Ag_1, Ag_2, \phi))$ iff
 (i) $\exists j < i : M, \langle s_j, P \downarrow s_j \rangle \models \text{Create}(Ag_1, SC^P(id, Ag_1, Ag_2, \phi))$ &
 (ii) $M, \langle s_j, P \downarrow s_j \rangle \models \neg\phi$ &
 (iii) $\exists P' \in \sigma^{s_i} : P' \downarrow s_j \in \mathbb{R}_{scp}(s_j, Ag_1, Ag_2)$
-

TABLE VII
SEMANTICS OF VIOLATION ACTION

Example 4: When *Cus* pays for the goods, but *Mer* does not deliver them within a specified time, then *Mer* violates his commitment. Through the path $\langle s_2, s_3, s_5, s_8, \dots \rangle$ the content *delivergoods* is false.

Release action: the semantics of release action (see Table VIII) is similar to the semantics of withdrawal. The only difference is that the release action is performed by the creditor while withdrawal is performed by the debtor (see Fig. 3).

-
- M20.** $M, \langle s_i, P \rangle \models \text{Release}(Ag_2, SC^P(id_1, Ag_1, Ag_2, \phi))$ iff
 (i) $\exists j < i : M, \langle s_j, P \downarrow s_j \rangle \models \text{Create}(Ag_1, SC^P(id_1, Ag_1, Ag_2, \phi))$ &
 (ii) $P \downarrow s_j \notin \mathbb{R}_{scp}(s_j, Ag_1, Ag_2)$ &
 (iii) $\exists P' \in \sigma^{s_i} : P' \downarrow s_j \in \mathbb{R}_{scp}(s_j, Ag_1, Ag_2)$
-

TABLE VIII
SEMANTICS OF RELEASE ACTION

Example 5: The customer *Cus*, before paying for the goods, can release the offer. Thus, no accessible path exists between *Cus* and *Mer* from s_4 . However, an accessible path still exists from s_3 .

Assignment action: the semantics of assignment action of a propositional commitment (see Table IX) is satisfied in the model M at s_i along path P iff (i) the creditor Ag_2 releases the current commitment at s_i through P ; and (ii) a new commitment with the same debtor and a new creditor appears at s_i , so that the formula $SC^P(id_1, Ag_1, Ag_3, \phi')$ is true at $M, \langle s_i \rangle$.

-
- M21.** $M, \langle s_i, P \rangle \models \text{Assign}(Ag_2, Ag_3, SC^P(id_0, Ag_1, Ag_2, \phi))$ iff
 (i) $M, \langle s_i, P \rangle \models \text{Release}(Ag_2, SC^P(id_0, Ag_1, Ag_2, \phi))$ &
 (ii) $\exists j < i : M, \langle s_j \rangle \models SC^P(id_0, Ag_1, Ag_2, \phi)$ &
 $M, \langle s_i \rangle \models SC^P(id_1, Ag_1, Ag_3, \phi')$ such that:
 (1) $\forall P' \in \sigma^{s_i}, M, \langle s_j, P' \downarrow s_j \rangle \models \phi \Leftrightarrow M, \langle s_i, P' \rangle \models \phi'$ &
 (2) $\mathbb{F}(\phi) \equiv \mathbb{F}(\phi')$
-

TABLE IX
SEMANTICS OF ASSIGNMENT ACTION

The most important issue in this semantics is that the content ϕ' of the new commitment is not necessarily the same as for the assigned one (ϕ), but there is a logical relationship between them. This is because the second commitment appears after

the previous one. Thus, we need to consider the temporal component specifying the deadline of the first commitment. The logical relationship between ϕ and ϕ' is as follows: (1) ϕ' is true at the current state s_i through a given path P' iff ϕ is true at s_j where the original commitment has been created through the prefix $P' \downarrow s_j$; and (2) the two contents are logically equivalent when the temporal operators are removed. We consider the current state s_i in (1) because the new content ϕ' should be true starting from the moment where the new commitment is established (see Fig.4).

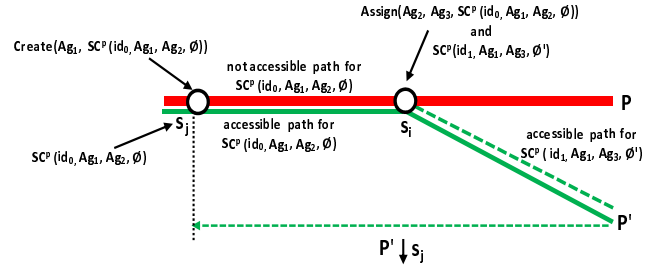


Fig. 4. Assign action at the state s_i along the path P

To clarify this issue, let us suppose that the content of the assigned commitment is $\phi = X^+X^+p$ where p is an atomic proposition and the assignment action takes place at the next moment after the creation action. The content of the resulting commitment should be then $\phi' = X^+p$, which is the content we obtain by satisfying the conditions (1) and (2). By (1) we have X^+p is true at a state s_i through a path P' iff X^+X^+p is true at the state s_j ($s_j = s_{i-1}$) through $P' \downarrow s_j$; and by (2) we have $\mathbb{F}(X^+X^+p) \equiv \mathbb{F}(X^+p)$. The second condition is added to guarantee that the relationship between the contents is not arbitrary.

Example 6: Suppose *Cus* commits to pay 200 to *Mer* in two days. After one day, *Mer*, for some reasons, assigns this commitment to *Mer*₁ (we suppose that there is an agreement between *Cus* and *Mer*₁ about this commitment). Thus, *Mer* releases the commitment with *Cus* and a new commitment between *Cus* and *Mer*₁ is established to pay the 200 after only one day.

In the semantics proposed in previous frameworks (for example in [13], [14] and [20]), the two commitments have the same content, which implicitly suppose that the creation of the commitment and its assignment take place at the same moment. The previous example cannot be managed using this assumption.

Delegation action: the semantics of delegation action (see Table X) is similar to the semantics of assignment. The only difference is that delegation is performed by the debtor while assignment is performed by the creditor. Therefore, instead of release, the semantics is defined in terms of withdraw.

Example 7: Suppose *Cus* commits to pay 200 to *Mer*

in two days. After one day, *Cus*, for some reasons, delegates this commitment to a financial company (*Bank*) to pay the 200 to *Mer* on his behalf. Thus, *Cus* withdraws his commitment and a new commitment between *Bank* and *Mer* is established to pay the 200 after only one day.

M22. $M, \langle s_i, P \rangle \models \text{Delegate}(Ag_1, Ag_3, SC^p(id_0, Ag_1, Ag_2, \phi))$ iff
 (i) $M, \langle s_i, P \rangle \models \text{Withdraw}(Ag_1, SC^p(id_0, Ag_1, Ag_2, \phi))$ &
 (ii) $\exists j < i : M, \langle s_j \rangle \models SC^p(id_0, Ag_1, Ag_2, \phi)$ &
 $M, \langle s_i \rangle \models SC^p(id_1, Ag_3, Ag_2, \phi')$ such that:
 (1) $\forall P' \in \sigma^{s_i}, M, \langle s_j, P' \downarrow s_j \rangle \models \phi \Leftrightarrow M, \langle s_i, P' \rangle \models \phi'$ &
 (2) $\mathbb{F}(\phi) \equiv \mathbb{F}(\phi')$

TABLE X
SEMANTICS OF DELEGATION ACTION

IV. COMMITMENTS' PROPERTIES

The aim of this section is to prove that the model aforementioned in the previous section presents a satisfactory “logic of commitment”. We show some of desirable properties related to the semantics of actions on social commitments that are fundamental for soundness considerations where alignment [7] among interacting agents in distributed systems is satisfied. In the rest of this paper, the set of all models is denoted \mathcal{M} .

Proposition 1: *If a commitment is created, then it has been never created before.*

$$AG^+ [Create(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))] \Rightarrow X^- G^- \neg Create(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))$$

Proof: Let M be a model in \mathcal{M} , s_i be a state in \mathbb{S} , and P be a path in σ . Also, suppose that:

$$M, \langle s_i, P \rangle \models Create(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))$$

(Semantics of creation action)

$$\Rightarrow \forall j < i \ M, \langle s_j, P \rangle \models \neg SC^p(id, Ag_1, Ag_2, \phi)$$

(Semantic calculus)

$$\Rightarrow M, \langle s_i, P \rangle \models X^- G^- \neg SC^p(id, Ag_1, Ag_2, \phi) \quad (1)$$

Let us now suppose that:

$$M, \langle s_i, P \rangle \models X^- F^- Create(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))$$

(Semantics of creation action)

$$\Rightarrow M, \langle s_i, P \rangle \models X^- F^- SC^p(id, Ag_1, Ag_2, \phi)$$

There is then contradiction with (1). Consequently:

$$M, \langle s_i, P \rangle \models X^- G^- \neg Create(Ag_1, SC^p(id, Ag_1, Ag_2, \phi)) \quad \blacksquare$$

As a direct consequence of this proposition, we have the following lemma:

Lemma 1: *Once created, a commitment cannot be created again in the future.*

$$AG^+ [Create(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))] \Rightarrow X^+ AG^+ \neg Create(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))$$

Proof: Let us suppose that the negation is true. Therefore:

$$EF^+ [Create(Ag_1, SC^p(id, Ag_1, Ag_2, \phi)) \wedge \neg (X^+ AG^+ \neg Create(Ag_1, SC^p(id, Ag_1, Ag_2, \phi)))]$$

(Semantic calculus)

$$\Rightarrow EF^+ [Create(Ag_1, SC^p(id, Ag_1, Ag_2, \phi)) \wedge X^+ EF^+ Create(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))]$$

Consequently, there is a contradiction with Proposition 1 as the second creation cannot take place since there is a creation in its past. Thus, we are done. \blacksquare

Proposition 2: *Once withdrawn, a commitment cannot be withdrawn again in the future.*

$$AG^+ [Withdraw(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))] \Rightarrow X^+ AG^+ \neg Withdraw(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))$$

Proof: Let M be a model in \mathcal{M} , s_i be a state in \mathbb{S} , and P be a path in σ . Also, suppose that:

$$M, \langle s_i, P \rangle \models Withdraw(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))$$

(Semantics of withdrawal and creation actions)

$$\Rightarrow \exists j < i : M, \langle s_j, P \downarrow s_j \rangle \models SC^p(id, Ag_1, Ag_2, \phi)$$

$$P \downarrow s_j \notin \mathbb{R}_{scp}(s_j, Ag_1, Ag_2) \quad (2)$$

Let us now suppose that:

$$M, \langle s_i, P \rangle \models X^+ EF^+ Withdraw(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))$$

(Semantics of withdrawal action, X^+ and EF^+)

$$\Rightarrow \exists k > i \ \exists P' \in \sigma^{s_k} : P' \downarrow s_j \in \mathbb{R}_{scp}(s_j, Ag_1, Ag_2)$$

There is then contradiction with (2) because P' is a suffix of P . Consequently:

$$M, \langle s_i, P \rangle \models \neg X^+ EF^+ Withdraw(Ag_1, SC^p(id, Ag_1, Ag_2, \phi)) \Rightarrow M, \langle s_i, P \rangle \models X^+ AG^+ \neg Withdraw(Ag_1, SC^p(id, Ag_1, Ag_2, \phi)) \quad \blacksquare$$

In the same way, we can prove the following three propositions (3,4 and 5) using the semantics of withdraw, fulfillment, violation and release actions, X^+ and EF^+ .

Proposition 3: *Once withdrawn, a commitment cannot be fulfilled in the future.*

$$AG^+ [Withdraw(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))] \Rightarrow X^+ AG^+ \neg Fulfill(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))$$

Proposition 4: *Once withdrawn, a commitment cannot be violated in the future.*

$$AG^+ [Withdraw(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))] \Rightarrow X^+ AG^+ \neg Violate(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))$$

Proposition 5: *Once withdrawn, a commitment cannot be released in the future.*

$$AG^+ [Withdraw(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))] \Rightarrow X^+ AG^+ \neg Release(Ag_2, SC^p(id, Ag_1, Ag_2, \phi))$$

Proposition 6: *Once withdrawn, a commitment cannot be assigned in the future.*

$$AG^+ [Withdraw(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))] \Rightarrow \forall Ag \in AGT \ X^+ AG^+ \neg Assign(Ag_1, Ag, SC^p(id, Ag_1, Ag_2, \phi))$$

Proof: Let M be a model in \mathcal{M} , s_i be a state in \mathbb{S} , and P be a path in σ . Also, suppose that:

$$M, \langle s_i, P \rangle \models Withdraw(Ag_1, SC^p(id, Ag_1, Ag_2, \phi))$$

(Proposition 5)

$$\Rightarrow M, \langle s_i, P \rangle \models X^+ AG^+ \neg Release(Ag_2, SC^p(id, Ag_1, Ag_2, \phi))$$

(Semantics of assignment action)

$\Rightarrow \forall Ag \in \mathcal{AGT}$
 $M, \langle s_i, P \rangle \models X^+AG^+ \neg Assign(Ag_2, Ag, SC^P(id, Ag_1, Ag_2, \phi))$ ■

Using Proposition 2 and the semantics of delegation action, we can prove the following proposition:

Proposition 7: *Once withdrawn, a commitment cannot be delegated in the future.*

$$AG^+[Withdraw(Ag_1, SC^P(id, Ag_1, Ag_2, \phi)) \Rightarrow \forall Ag \in \mathcal{AGT} \\ X^+AG^+ \neg Delegate(Ag_1, Ag, SC^P(id, Ag_1, Ag_2, \phi))]$$

As for previous propositions for withdrawal action, we can also prove the following proposition for fulfillment action:

Proposition 8: *Once fulfilled, a commitment cannot be fulfilled again, withdrawn, violated, released, assigned or delegated in the future.*

$$AG^+[Fulfill(Ag_1, SC^P(id, Ag_1, Ag_2, \phi)) \Rightarrow \forall Ag \in \mathcal{AGT} \\ X^+AG^+ \neg [Fulfill(Ag_1, SC^P(id, Ag_1, Ag_2, \phi)) \\ \vee Withdraw(Ag_1, SC^P(id, Ag_1, Ag_2, \phi)) \\ \vee Violate(Ag_1, SC^P(id, Ag_1, Ag_2, \phi)) \\ \vee Release(Ag_2, SC^P(id, Ag_1, Ag_2, \phi)) \\ \vee Assign(Ag_2, Ag, SC^P(id, Ag_1, Ag_2, \phi)) \\ \vee Delegate(Ag_1, Ag, SC^P(id, Ag_1, Ag_2, \phi))]]$$

V. DISCUSSION AND RELATED WORK

A. Discussion

Our logical model is useful when developing agent communication languages (ACL) thanks to its foundation based on social semantics. Unlike mentalistic semantics that specifies the semantics of communicative acts in terms of pre- and post-conditions contingent on so-called agent's mental states (e.g. beliefs, desires and intentions), this social semantics can be verified [4], [21], [22]. This is because our semantics allows for tracing the status of existing commitments at any point in time given observed actions. In commitment protocols, social commitments capture a high meaning of interactions and provide a useful level of abstraction. In this sense, Yolum and Singh [24] have used commitment operations to show how to build and execute commitment protocols and how to reason about them using event calculus. In the same way, Mallya and Singh [14] have showed how to reason about subsumption among commitment protocols and how to refine and aggregate protocols based on commitment semantics and operations. Also, Desai and Singh [9] have studied a composition of commitment protocols and concurrent operations. Our proposal belongs to the same line of research and can be used to specify commitment protocols in terms of creation and manipulation commitments using accessibility relation and the principle properties of commitment.

In fact, our framework provides a unified semantics of social commitments and associated operations that can be used to enhance, e.g., Tropos methodology [16], where Tropos is an agent-oriented software methodology, via capturing the meaning of interactions in terms of task dependencies among

communicating agents. Also, in [20] Singh has delineated the model-theoretic semantics of commitments by postulating some rules as ways of using and reasoning with commitments. This model combines two commitments (practical and dialogical), in the sense that when a commitment arises within an argument and the content is satisfied with the same argument, then practical commitment would be satisfied. However, this model does not include the semantics of commitment operations. Chopra and Singh [7] have used the theoretical model proposed in [20] to study the semantics of commitment operations with message patterns that implement commitment operations with some constraints on agents' behaviors to tackle the problem of autonomy in distributed systems. This semantics is expressed in terms of the set of propositions that can be inferred from the observation sequence that agents sent or received. Moreover, this semantics must correspond to the postulates introduced in [20]. However, the formal language of those postulates is based on enhancing *LTL* (linear-time logic) with two commitment modalities. Thus, this language is less expressive than the formal language introduced here, which is more compatible with agent choices. Furthermore, our semantics is based on Kripke structure with accessibility relations, which enables us to prove that the proposed model is computationally grounded [23] and to verify this semantics. Finally, in [5], the propositional commitment is fulfilled when the creditor does not believe that the commitment's content is false and he cannot challenge it anymore. However, this semantics uses mental states which cannot be verified. The semantics defined here for conditional commitments is different from the semantics defined in [17] and [20]. In [17], conditional commitments are considered as intentions, while commitments as social notion are different from private intentions. In [20], Singh models conditional commitments as fundamental and unconditional commitments as special cases where the antecedent is true. In our semantics, the conditional commitments are transformed into propositional commitments in all accessible paths where the underlying condition is true.

B. Related Work

Let us now focus on comparing the proposed semantics for commitment operations with the related ones. The semantics proposed here is close to the semantics introduced in [3], but does not suffer from the "recursion" problem, which is the main problem in [3]. Recursion means the semantics of one operation depends on the semantics of one or more other operations. For example, in [3], a propositional commitment is satisfied along a path P at a state s_i iff it is active in this state along this path, and it was already created at a state s_j , and along this path from the state s_j the commitment content is true. Also, a commitment is active iff this commitment was already created and until the current moment the commitment was not withdrawn. Consequently, the model checking technique for this logic is very complex and probably suffers from the *state explosion* problem in the early phases. On the contrary, the semantics we presented here is independent, for each operation, of the semantics of

other operations. Furthermore, our semantics of commitment operations is different from the ones given in [6], [8], [13] and [14]. Particularly (as discussed in Section III-B3), assignment and delegation operations should consider that the content of the new resulting commitment could be different from, and has a logical relationship with the content of the assigned and delegated commitment. This issue is not captured in previous frameworks. In addition, unlike our semantics, the violation operation has been disregarded.

C. Directions: Theoretical and Practical

Here, we outline two promising directions of future work.

Theoretically. We plan to improve our proposed semantics by removing the simplification based on the supposition that actions are only momentary and considering time frames between the execution of actions. We also plan to study the commitment operations needed to handle meta-commitments, that is commitments about commitments, that often arise in real-life scenarios. The proposed semantics would be augmented with argumentation to enhance the Tropos methodology, which we plan to apply for modeling and establishing communities of web services introduced in [12].

Practically. We intend to integrate our logical model with the logic of agent programs developed in [1] for the implementation of agents. Furthermore, a rigorous semantics opens up the way for improving the verification of logic-based protocols that govern a set of autonomous interacting agents against given properties. The mainstream step in this regard would be to map the commitment semantics introduced here to conventional verification technologies. Our semantics is based on Kripke structures (like interpreted systems). Currently model checking techniques work best for logics whose semantics is given via accessibility relations with the extension of CTL^* as proposed in [4]. Two complementary software tools are suggested to implement model checking algorithms to verify whether or not the model M satisfies the proposed commitments' properties (i.e., $M \models \phi$). Model checking algorithm for our logic can be implemented via *NuSMV* tool, which is the best-known for CTL^* . On the other hand, the proposed logic and associated properties, which need to be checked, can be specified as tableau-based rules. Such rules provide a simple decision procedure for the logic and overcome model checking algorithm from *state explosion* problem. As proposed in [4], the verification method could be based on the translation of formulae into a variant of alternating tree automata called Büchi tableau automata (ABTA).

ACKNOWLEDGEMENTS

We would like to thank NSERC (Canada), NATEQ and FQRSC (Québec) for their financial support.

REFERENCES

- [1] N. Alechina, M. Dastani, B.S. Logan and J.-J. Ch. Meyer. A Logic of Agent Programs. In Proc. of the Twenty-Second AAAI Conf. on Artificial Intelligence (AAAI), pp.795-800, 2007.
- [2] J. Bentahar, B. Moulin, J.-J. Ch. Meyer and B. Chaib-draa. A Logical Model for Commitment and Argument Network for Agent Communication. In Proc. of the Int. Joint Conf. on AAMAS, pp.792-799, 2004.
- [3] J. Bentahar, B. Moulin, J.-J. Ch. Meyer and Y. Lespérance. A New Logical Semantics for Agent Communication. K. Inoue, K. Satoh, F. Toni (eds.), Computational Logic in Multi-Agent Systems. LNAI 4371, pp.151-170, 2007.
- [4] J. Bentahar, J.-J. Ch. Meyer and W. Wan. Model Checking Communicative Agent-based Systems. In Knowledge-Based Systems, Special Issue on Intelligent Software Design, vol.22(3), Elsevier, pp.142-159, 2008.
- [5] G. Boella, R. Damiano, J. Hulstijn and L. Torre. Distinguishing Propositional and Action Commitment in Agent Communication. In Proc. of the Workshop on Comput. Modles of Natural Argument (CMNA'07).
- [6] A.K. Chopra and M.P. Singh. Constitutive Interoperability. In Proc. of the Int. Joint Conf. on AAMAS, pp.797-804, 2008.
- [7] A.K. Chopra and M.P. Singh. Multiagent Commitment Alignment. In Proc. of the 8th Int. Joint Conf. on Autonomous Agents and MultiAgent Sys. (AAMAS), May 2009.
- [8] A.K. Chopra and M.P. Singh. Nonmonotonic Commitment Machines. In Dignum F. (eds) Advances in Agent Communication. LNAI 2922, pp.183-200, 2004.
- [9] N. Desai, A.K. Chopra and M.P. Singh. Representing and Reasoning About Commitments in Business Processes. In Proc. of the 22nd Conf. AAAI, pp.1328-1333, 2007.
- [10] E.A. Emerson and J.Y. Halpern. Sometimes and not never, Revisited: on Branching versus Linear time Temporal Logic. Journal ACM (JACM) vol.33(1), pp.151178, 1986.
- [11] N. Fornara and M. Colombetti. Operational Specification of a Commitment-based Agent Communication Language. In Proc. of the Int. Joint Conf. on AAMAS, pp.535-542, 2002.
- [12] Z. Maamar, S. Subramanian, J. Bentahar, P. Thiran P and D. Benslimane. An Approach to Engineer Communities of Web Services Concepts, Architecture, Operation and Deployment. In the Int. Journal of E-Business Research, vol.5(4), IGI Global.
- [13] A.U. Mallya, P. Yolum and M.P. Singh. Resolving Commitments among Autonomous Agents. F. Dignum (eds.), Advances in Agent Communication. LNAI 2922, pp.166-182, 2004.
- [14] A.U. Mallya and M.P. Singh. An Algebra for Commitment Protocols. Autonomous Agents and Multi-Agent Systems, vol.14(2), pp.143-163, 2007.
- [15] M. El-Menshawy, J. Bentahar and R. Dssouli. A New Semantics of Social Commitments using Branching Space-Time Logic. In Proc. of the IEEE/WIC/ACM Inter. Conf. on Inte. Agent Technology (IAT'09), Logics for Inte. Agents and Multi-Agent Systems, Milano, Italy, September 15-18, 2009.
- [16] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia and J. Mylopoulos. Tropos: An Agent-Oriented Software Development Methodology. Autonomous Agents and Multi-Agent Systems, vol.8(3), pp.203-236, 2004.
- [17] M.K. Shakil and L. Yves. On the Semantics of Conditional Commitment. In Proc. of the Int. joint Conf. on AAMAS, pp.1337-1344, 2006.
- [18] M.P. Singh. An Ontology for Commitments in Multiagent Systems: toward a Unification of Normative Concepts. AI and Law, vol.7, pp.97113, 1999.
- [19] M.P. Singh. A Social Semantics for Agent Communication Languages. Issues in Agent Communication, Dignum, F. and Greaves, M. (eds.), LNAI 1916, pp.31-45, 2000.
- [20] M.P. Singh. Semantical Considerations on Dialectical and Pratical Commitments. In Proc. of the 23rd Conf. AAAI, pp. 176-181, 2008.
- [21] M.P. Singh. Agent Communication Languages: Rethinking the Principles. IEEE Computer, vol.31(12), pp.40-47, 1998.
- [22] P. Torroni, F. Chesani, P. Yolum, M. Gavanelli, M.P. Singh, E. Lamma, M. Alberti and P. Mello. Modelling Interactions via Commitments and Expectations. In Handbook of Research on Multi-Agent Systems: Semantics and Dynamics of Organizational Models, ch.11, V. Dignum (eds.), pp.263-284, 2009.
- [23] M. Wooldridge. Computationally Grounded Theories of Agency. In E. Durfee, editor, In Proc. of the 4th Int. Conf. on Multi-Agent Sys. (ICMAS 2000). IEEE Press, 2000.
- [24] P. Yolum and M.P. Singh. Flexible Protocol Specification and Execution: Applying Event Calculus Planning using Commitments. In Proc. of the Int. Joint Conf. on AAMAS, pp.527-534, 2002.
- [25] M. Verdicchio and M. Colombetti. A Logical Model of Social Commitment for Agent Communication. In Proc. of the Int. Joint Conf. on AAMAS, pp.528-535, 2003.

Externalisation and Internalization: A New Perspective on Agent Modularisation in Multi-Agent System Programming

Alessandro Ricci
DEIS, University of Bologna
Cesena, Italy
Email: a.ricci@unibo.it

Michele Piunti
DEIS, University of Bologna
Cesena, Italy
Email: michele.piunti@unibo.it

Mirko Viroli
DEIS, University of Bologna
Cesena, Italy
Email: mirko.viroli@unibo.it

Abstract—Agent modularisation is a main issue in agent and multi-agent system programming. Existing solutions typically propose some kinds of constructs – such as *capabilities* – to group and encapsulate in well-defined modules *inside* the agent different kinds of agent features, that depend on the architecture or model adopted—examples are goals, beliefs, intentions, skills. In this paper we introduce a further perspective, which can be considered complimentary to existing approaches, which accounts for *externalizing* some of such functionalities into the computational environment where agents are (logically) situated. In this perspective, agent modules are realised as suitably designed *artifacts* that agents can dynamically exploit as *external* tools to enhance their action repertoire and – more generally – their capability to execute tasks. Then, to let agent (and agent programmers) exploit such capabilities abstracting from the low-level mechanics of artifact management and use, we exploit the dual notion of *internalization*, which consists in dynamically consulting and automatically embedding high-level usage protocols described in artifact *manuals* as agent plans. The idea is discussed providing some practical examples of use, based on CArtAgO as technology for programming artifacts and Jason agent platform to program the agents.

I. INTRODUCTION

Agent modularisation is a main issue in agent-oriented software engineering and multi-agent system (MAS) programming, accounting for devising proper structures and mechanisms to modularise agent behaviour, enhancing maintainability, extensibility and reuse of agent-based software. Existing solutions – which are briefly surveyed in Section IV – typically propose constructs that make it possible to group, encapsulate and reuse in well-defined modules agent features, that can vary according to the architecture or model adopted: for instance, modularisation in BDI agents have been proposed in terms of capabilities [3], [2], goals [17], intentions [8], to mention some.

In all existing approaches modules are components *inside* agents. In this paper we introduce a further complimentary perspective, which accounts for improving modularity by *externalizing* some functionalities into the *computational environment* where agents are (logically) situated, as external facilities that agents exploit as tools extending their capabilities.

The background of this idea is given by the research work on *environment design and programming* in MAS [19], [15], in which the computational environment where agents are situated is considered a *first-class abstraction* that can be suitably designed and programmed so as to improve MAS engineering, encapsulating functionalities that concern, for instance, agent interactions, coordination and organisation.

In this context, CArtAgO [15], [16] – which will be exploited in this paper – has been proposed as a general-purpose framework and infrastructure for building shared computational worlds that agents, possibly belonging to heterogeneous agent platforms and written using different agent programming languages [14], can exploit to work together. Being based on the A&A (Agents and Artifacts) meta-model [15], [12], [16], CArtAgO's computational environments are modelled as set of distributed workspaces, containing dynamic sets of *artifacts*.

The artifact abstraction is a key concept on which is the contribution of this paper is based. From the agent viewpoint, artifacts are first-class entities of agents' world, representing resources and tools that agents can dynamically instantiate, share and use to support individual and collective activities. From the MAS designer viewpoint, artifacts are useful to uniformly design and program those abstractions inside a MAS that are not suitably modelled as agents, and that encapsulate functions to be exploited by individual agents or the overall MAS—for instance mediating and empowering agent interaction and coordination, or wrapping external resources. CArtAgO provides a concrete computational and programming model for artifacts [15], composed by a set of Java-based API to program artifacts on the one side, and agent API to work inside artifact-based environment on the other side.

The availability of artifact-based computational environments in multi-agent system programming makes it possible to enrich the strategies for modularising agents by exploiting artifacts as modules that can be dynamically instantiated/used/composed, extending the basic repertoire of agent actions and capabilities. So, instead of being wrapped into modules inside agents – either structuring the agent program

or extending the agent architecture – such capabilities are *externalised* into artifacts that agents can use and exploit as personal – and in some cases shared – tools.

In this paper we develop this idea, providing some practical examples using **CARTAgO** and **Jason** agent programming language. It is important to remark that this approach is not meant to replace existing proposals, but to be integrated with them. On the one side, some agent features are clearly not externalisable, or – at least – it is not useful to externalise them. For instance, for cognitive agents, capabilities concerning deliberation or the manipulation of the internal mental state. On the other side, the approach allows for properties which are not typically provided by existing proposals. For instance, the reuse of the same kind of module (artifacts) across different agent programming languages and platforms.

The remainder of the paper is organised as follows: in Section II we describe in detail the idea, providing some examples to clarify the approach. Then, in Section III we introduce *internalization* as a key mechanism layered on top of externalisation that allows agents and agent programmers to exploit functionalities externalized in artifacts abstracting as much as possible from the low-level mechanics of artifact management and use. In Section IV we provide an overview of existing works on agent modularisation and how the contribution of this paper is related to them. Finally, in Section V we provide concluding remarks, sketching current limitations and the next steps planned to further develop of the idea.

II. EXTERNALISATION: AGENT MODULES IMPLEMENTED AS ARTIFACTS

A. The Basic Idea

The basic idea is to exploit artifacts as modules to encapsulate new capabilities for agents, in particular extending the repertoire of agent actions with the set of operations provided by artifacts¹. We call this *externalisation* since the capabilities of an agent are not extended by acting agent internal architecture or program, but by extending the set of external resources and tools (artifacts) that the agent can use to do its work.

By applying externalisation, a module is conceived as a tool that the agent may eventually create and use by need. In particular: artifact operations encapsulate the functionalities that would be provided by module actions; artifact usage interface and observable properties (and events) represent the module interface; the non-observable state of the artifact is used to implement the hidden inner state of the module; and finally the *manual* of an artifact can be used to store the description of high-level usage protocols accompanying the module—this point will be discussed in detail in Section III.

¹the main features of the artifact abstraction are extensively described in [12], [16], [15]. Briefly, each artifact has a usage interface listing a set of usage interface controls that can be used to trigger and control the execution of operations inside the artifact. By executing operations, an artifact can generate observable events (signals) that can be perceived both by the agent using the artifact and by all those that are focussing (observing) it. Besides observable events, an artifact can have observable properties, whose value (and changes) are automatically perceived by all the observing agents

At runtime (execution time) the burden of the execution of modules is no more on the agent side, like in the case of modules implemented as components inside agents, but on the artifact side: artifact operations are executed asynchronously by independent control flows, managed by the **CARTAgO** machinery. The agent can control operations execution by means of the usage interface of the artifact, perceiving its state and results in terms of observable properties and events. This has a strong impact on efficiency at runtime: (a) agents do not waste time and computational resources for the execution of the processes related to the module functionalities; (b) the approach transparently exploits the concurrency and parallelism support provided by the underlying execution environment.

Then, module management is mapped on to artifact creation/disposal/discovery, in particular module activation is given by instantiating the artifact or by locating an existing one; module composition is realised by using multiple artifacts. Actually, the approach supports also a kind of module *sharing* by exploiting artifacts shared and co-used simultaneously by multiple agents: this can be very useful for supporting effective and efficient forms of agent coordination (Subsection II-E).

In the following, we clarify and discuss the idea by describing some concrete examples of increasing complexity, exploiting **CARTAgO** to implement artifacts and **Jason** [1] to program agents exploiting artifacts as modules. It's worth noting that here we use **Jason**, but an analogous discussion would be for other agent programming languages, such as **2APL**, or platforms like **Jadex**. The examples are classified along two basic dimensions: the state dimension – differentiating between state-less and state-full modules (tools) – and the sharing dimension – differentiating between personal and shared modules (tools).

B. Modules as State-less Tools

The simplest kind of module is given by a library of *internal actions* which are meant to be exploited by agents as functions, extending the basic computing capabilities provided by default by the agent language.

As a concrete example, suppose to extend agents with some math capabilities not directly supported by the language—let's take the *sine* function as a simple example. Languages like **Jason** allow for solving the problem by extending the agent architecture, with the implementation of new internal actions exploiting the Java-based API provided by the **Jason** platform. Externalisation makes it possible to solve the problem without the need of extending directly agents, by programming a new kind of artifact – functioning as a *calculator* tool in this case – that the agent can instantiate and (re-)use by need. Fig. 1 shows a sketch of its implementation in **CARTAgO** API and of a **Jason** agent exploiting the functionality². The action module `sin(+Value, ?Result)` is implemented by the `computeSin(+Value)` operation of the artifact, and

²Details about the artifact abstraction and **CARTAgO** API, as well as **Jason** and their integration, are outside the scope of this paper: the interested reader can find them in literature [15], [12], [16], [14], [1]


```

package tools;

public class Calculator extends Artifact {

    @OPERATION void computeSin(double x){
        signal("sin",x,Math.sin(x));
    }
    @OPERATION void computeCos(double x){
        signal("cos",x,Math.cos(x));
    }
    @OPERATION void computeSqrt(double x){
        if (x >= 0){
            signal("sqrt",x,Math.sqrt(x));
        } else {
            signal("math_error");
        }
    }
    ...
}

// Jason agent using its calculator

!doComputations.

+!doComputations
  <- ?mytool("tools.Calculator", Id);
  cartago.use(Id,computeSin(1.57),s0);
  cartago.sense(s0,sin(1.57,Y));
  cartago.use(console,
    println("The sin value of 1.57 is ",Y)).

+?mytool(ToolType, Id)
  <- .my_name(AgName);
  .concat(AgName,"-",ToolType,ToolName);
  cartago.makeArtifact(ToolName,ToolType,Id);
  +mytool(ToolType, Id) .

```

Fig. 1. (Left) A Calculator artifact encapsulating math functionalities. `computeSin` operation, once triggered, generates an observable event of the type `sin(X,Y)` which is then perceived by the agent using the calculator. (Right) A Jason agent exploiting the calculator: the first time the calculator is used it is created, using a conventional name given by concatenation of the agent name and the artifact type.

action execution is realised in terms of a sequence of basic `CARTAGO` actions to interact with it. In particular, the agent first retrieves the tool identifier – eventually creating the artifact if it is the first time it is used and keeping track of such identifier by a `mytool` belief; then, it triggers the execution of the operation on the tool (by means of the `use CARTAGO` primitive) and then exploits a sensor to perceive the result (by means of the `sense CARTAGO` primitive). The result is represented by an observable event `sin(X,Y)` generated by the `signal` primitive executed in the `computeSin` operation. It's worth remarking that the computation of the sine value is done asynchronously w.r.t. the agent activities: synchronisation occurs when the agent inspects the sensor to perceive the result.

This first example – in spite of its simplicity – is useful to give a taste of the approach: instead of extending the agent architecture by means of new (internal) `sin` action, in this case the extension is achieved by means of an external calculator tool that an agent can instantiate and use. Being externalised into an artifact, the functionalities can be exploited by any kind of agent whose platform has been integrated with `CARTAGO`—besides Jason, other examples include `Jadex`, `2APL`, `AgentFactory`: so the approach promotes extensibility and reusability across heterogeneous agent languages and platforms.

C. Modules as State-ful Tools

Then, besides state-less modules, artifacts can be useful to encapsulate functionalities involving a state and providing actions working with such state, functioning as personal state-ful tools. The state can be either an observable part, i.e. that can be considered part of the agent knowledge, or a hidden part of the module. The observable part is mapped onto artifact observable properties, which are then perceived by the agent observing the artifact as percepts (mapped into beliefs in cognitive architectures).

As an example, consider the `Calculator2` artifact depicted in Fig. 2, providing a support for executing a sequence of operations, keeping track and making it observable the

updated result by means of the `result` observable property and providing functionalities to undo the operations. Fig. 2 shows an example of an agent using the calculator, adding repeatedly a value (3.0) by “pressing the button” `add` until it perceives that the result is greater than 10. After that, it restores the previous result (which is 9, in this case) and prints it on the console.

This second example shows how the approach supports the management of observable information of the module on the one side and *information hiding* on the one side: inner structures needed to realise the module functionalities – such as the list of the partial results, to enable undo and redo in the example – are implemented by artifact inner data structures, which are accessed and changed by artifact operations.

D. Modules Wrapping External Actions

In previous examples we considered modules encapsulating sets of *internal* actions: besides these ones, modules can be also devised so as to extend agents with capabilities to access/interact with external resources (such as data-base), including resources to communicate with external systems (such as network channels, GUIs). In that case, the externalisation perspective accounts for implementing such modules as tools wrapping the access and interaction with those external resources, hiding as much as possible the low-level details related to the use of the resources and providing the agent a high-level interface for exploiting the functionalities. Actually `CARTAGO` provides – as an auxiliary library – a basic set of artifact types that can be exploited to this end, including tools for working with ODBC data-bases, for using socket-based network channels, and for creating and managing graphical user interfaces. Examples of this kind of tools can be found in `CARTAGO` distribution—not reported here for lack of space.

E. Modules Wrapping Mechanisms for Interaction, Coordination, Organisation

Agent coordination is a main issue in multi-agent system programming; direct communication models – including approaches based on speech-act based conversations – are not always the most effective solution to achieve agent coordination

```

public class Calculator2 extends Artifact {
    Stack<Double> results;

    @OPERATION void init(){
        defineObsProperty("result",0);
        results = new Stack<Double>();
    }
    @OPERATION void add(double x){
        double res = getObsProperty("result").doubleValue();
        results.push(res);
        updateObsProperty("result",res + x);
    }
    @OPERATION void sub(double x){
        double res = getObsProperty("result").doubleValue();
        results.push(res);
        updateObsProperty("result",res - x);
    }
    @OPERATION void undo(){
        if (!results.isEmpty()){
            updateObsProperty("result",results.pop());
        } else {
            signal("result_stack_empty");
        }
    }
}

```

```

!doComputations.

+!doComputations \
  <- ?mytool ("tools.Calculator2",Calc);
  cartago.focus (Calc);
  !doSums (Calc) .

+!doSums (Calc): result (X) & X<=10
  <- cartago.use (Calc,add(3.0),s0);
  cartago.sense(s0,op_exec_completed("add"));
  !doSums (Calc) .

+!doSums (Calc): result (X) & X>10
  <- cartago.use (Calc,undo,s0);
  cartago.sense(s0,op_exec_completed("undo"));
  cartago.observeProperty (Calc,result (Y));
  cartago.use (console,print ("Final value: ",Y)).

```

Fig. 2. (Left) State-full extension of the Calculator, exploiting observable properties. (Right) Jason agent exploiting the calculator.

```

public class Semaphore extends Artifact {
    int count;

    @OPERATION void init(int startCount){
        count = startCount;
    }

    @OPERATION(guard="isFree") void acquire(){
        count--;
    }

    @GUARD boolean isFree(){
        return count > 0;
    }

    @OPERATION void release(){
        count++;
    }
}

```

```

!doJob.

+!doJob
  <- !locateTool ("tools.Semaphore","cs",[1],Tool);
  !work(Tool) .

+!work(Tool)
  <- cartago.use (Tool,acquire);
  !doMyCriticalTask(0);
  cartago.use (Tool,release);
  !work(Tool) .

+!doMyCriticalTask(C) : C < 10
  <- .println(C); .wait(10); !doMyCriticalTask(C+1) .
+!doMyCriticalTask(10) .

+!locateTool (Type,Name,Args,Id) : not tool_avail (Name)
  <- cartago.lookupArtifact (Name,Id) .
-!locateTool (Type,Name,Args,Id) : not tool_avail (Name)
  <- +~tool_avail (ToolName); !locateTool (Type,Name,Args,Id) .
+!locateTool (Type,Name,Args,Id) : ~tool_avail (Name)
  <- cartago.makeArtifact (Name,Type,Args,Id) .
-!locateTool (Type,Name,Args,Id) : ~tool_avail (Name)
  <- -~tool_avail (Name); !locateTool (Type,Name,Args,Id) .

```

Fig. 3. (Left) A Semaphore artifact, that can be exploited as a tool for extending agents with basic synchronization capabilities. (Right) Jason agent exploiting the Semaphore for realising critical sections

and various kinds of interaction-oriented and coordination-oriented *mechanisms* can be devised to this end. From an agent programming language perspective, the implementation of these mechanisms typically accounts for extending the basic agent language with a specific set of primitives tailored to provide some kind of coordination/organisation functionalities. This strongly reminds *coordination languages* [7], which are orthogonal to classical computational languages (such as C, Java, Prolog) and extend them with basic coordination primitives to enable communication and synchronisation. Linda is a well-known example of coordination language [6]. Actually, in the context of multi-agent systems this occurs also for organisation: languages such as J-MOISE [9], for instance, extends the basic Jason language with MOISE organisational primitives.

This case is similar to the previous one, since such primitives can be considered external actions involving some kind of inter-actions with other agents. By adopting externalisation,

such capabilities can be encapsulated in proper artifacts, extending agents with coordination capabilities without the need of extending the agent language. Differently from the previous cases, these artifacts are meant to be *shared* by the agents – as a kind of shared modules – providing operations enabling and ruling the interaction among the agents exploiting them.

As a simple example, consider here the problem of extending an agent with the capability of executing *critical sections*, which require the coordination of all the agents running in the same environment. To this end, we can simply use a semaphore artifact functioning as a shared lock by the agents, providing two basic operations: to acquire it – to be used in the prologue of the critical section – and to release it – to be used in the epilogue of the section. Fig. 3 shows on the left the semaphore artifact and on the right an agent using it to realise a critical section. In this case all agents interact with the same artifact—called *cs* in the example. From an agent (and

```

usageprot compute_sin {
  :function sin(X,Y)
  :body {
    locateMyTool(ToolId);
    freshSensor(S);
    use(ToolId,computeSin(X),S);
    sense(S,sin(X,Y)).
  }
}

!doComputations
<- !setup;
!doTheJob.

+!doTheJob
<- cartago.consultManual("tools.Calculator");
cartago.consultManual("tools.Console").

+!doTheJob
<- !sin(1.57,Y);
!print("The sin value of 1.57 is ",Y).

```

Fig. 4. (Left) A usage protocol defined in the Calculator manual (Right) Jason agent exploiting the manual to use the Calculator

agent programmer) perspective, this can be seen as a facility extending the basic agent coordination capabilities, alternative to the use of communication protocols. The interested reader can find more complex examples of coordination tools in CArtaGO distribution: among the other the TupleSpace artifact, which in the perspective of this paper can be framed as a module extending agents with the Linda coordination language.

III. INTERNALIZATION: USING ARTIFACTS AS AGENT MODULES

Actually, a main problem of externalisation is the level of abstraction adopted for allowing an agent to access and exploit the new capabilities provided by the modules: when programming agents exploiting modules externalised into artifacts, the programmer must specify the details related to artifacts use and management. We tackle this problem by means of *internalization*.

Internalization accounts for introducing a proper abstraction layer which makes it possible to exploit artifacts functionalities in terms of agent actions, abstracting as far as possible – from an agent programmer point of view – from the low-level mechanics of artifact management and use. This can be achieved by exploiting artifacts *manual*. Being a feature of the basic artifact abstraction, the manual is that document providing a machine-readable description – written by the artifact developer – of artifact functionalities and operating instructions [12], [18]. Such information are meant to be dynamically read, interpreted and *internalized* by the agent, embedding such a knowledge in terms of proper *plans* about how to use the artifacts of that type and when.

Here we focus on the operating instructions, as that part of the manual describing *usage protocols*, i.e. high-level plans encapsulating sequences of low-level operations to be executed in order to exploit artifact functionalities. In current model, a usage protocol is characterised by a *function*³, which defines the functionality to be exploited, a *precondition*, defining the condition under which the functionality can be exploited, and a *body*, as a sequence of actions. Fig. 4 shows an example of usage protocol defined in the manual for the calculator, to exploit the sine function. A simple first-order logic-based language is used to define the protocols: the complete syntax and semantics of the language is not reported here for lack

of space, we describe the language informally by means of concrete examples.

The function is specified by means of `:function` tag and is represented by a logic term, possibly containing parameters detailing input and output (in terms of unbounded variables) information characterising the function. In the calculator example, `sin(X,Y)` is the function of the usage protocol to compute the sine function. The function of a usage protocol is directly linked to agent goals: a usage protocol with a function *func* is mapped into agent plan(s) that are triggered to achieve goals matching *func*, according to some kind of matching function that depends on the agent architecture adopted. In the case of Jason agents, for instance, the usage protocol is triggered to achieve goals of the type `sin(X,Y)`: in the example (Fig. 4, on the right) this happens by means of the `!sin(1.57,Y)` action.

The precondition can be specified by the `:precond` tag and is represented by a logic expression specifying the conditions that must hold concerning either the function parameters or agent beliefs⁴ (which typically can include the state of the observable properties of the artifact). If missing, the default value of the expression is `true`. Preconditions are used in the second example (Fig. 5), showing the manual of the semaphore artifact described in Subsection II-E, providing high-level usage protocols to execute critical sections. In particular, two alternative protocols are specified for entering a critical section, one to be used when the agent is not already inside the critical section and the other one in the opposite case. The belief `inside_cs(ToolId)` – added by one protocol when the entering succeeds – is used to distinguish this case.

The body – specified by means of the `:body` tag – contains a sequence of actions, including basic CArtaGO actions (`use`, `sense`, `focus`, etc.), auxiliary actions to locate artifacts and internal actions for inspecting and updating the belief and goal base of the agent. From a syntactical point of view, `;` is used as sequence operator, `+Bel` and `-Bel` is used to add and remove beliefs and `.` to indicate the end of the plan.

On the agent side, two further actions are provided respectively for consulting and forgetting the content of a manual, `consultManual(ArtifactTypeName)` and `forgetManual(ArtifactTypeName)`⁵. By consulting

⁴The notion of “belief” can be replaced here with “knowledge” for agent programming languages not having that concept

⁵the parameter does not refer to the name of a specific existing artifact, but to the name of an artifact type, which must be available by current workspace

³The term “function” here must be interpreted as “functionality”, so not related to functional programming languages or mathematical functions

```

usageprot enter_critical_section1 {
  :function enterCS
  :precond not inside_cs(_)
  :body {
    locateTool("Semaphore","cs",ToolId);
    use(ToolId,acquire); +inside_cs(ToolId).
  }}

usageprot enter_critical_section2 {
  :function enterCS
  :precond inside_cs(_)
  :body {}}

usageprot exit_critical_section {
  :function exitCS
  :precond inside_cs(ToolId)
  :body {
    use(ToolId,release); -inside_cs(ToolId).
  }}

!doJob
  <- !setup;
    !work.

+!setup
  <- cartago.consultManual("tools.Semaphore");
    cartago.consultManual("tools.Console").

+!work
  <- !enterCS;
    !doMyCriticalTask(0);
    !exitCS;
    !work(Tool).

+!doMyCriticalTask(C) : C < 10
  <- !println(C);
    .wait(10);
    !doMyCriticalTask(C+1).
+!doMyCriticalTask(10).

```

Fig. 5. (Left) Usage protocols defined in the Semaphore manual for doing critical sections (Right) Jason agent executing critical sections exploiting the usage protocols

the manual, the practical knowledge contained inside is fetched and translated into agent local plans, which are triggered by achievement goals which have the same signature of the function.

The key point here is that the agent programmer has not to be aware and explicitly code the usage protocol, which is specified – instead – by artifact developers: s/he must simply know the interface of the usage protocol, in terms of the function and beliefs involved. So the approach promotes a strong separation of concerns and finally more compact agent programs. This is exemplified by the source code of the Jason agent in Fig. 5, whose behaviour is analogous to the one in Subsection II-E but where `!enterCS` and `!exitCS` are the only lines of code that the agent programmer has to write to let the agent enter and exit a critical section.

IV. RELATED WORKS

Agent modularisation is a main open issue in agent programming languages and various solutions have been proposed in literature.

In [3], the notion of capability has been introduced and implemented in the JACK commercial Java-based multi-agent framework. Capabilities represent a cluster of components of a BDI agent, both encapsulating beliefs, events and plans and promoting global meta-level reasoning over them. From a software engineering perspective – which is the main perspective of this paper – capabilities enable software reuse, being building blocks that can be reused in different agents. This notion of capability is further refined and improved in *Jadex*, a Java and XML based BDI agent platform [2]. Capabilities are here generalised and extended so as to support an higher degree of reusability, devising a mechanism that allows for designing and implementing BDI agents as a composition of configurable agent modules (capabilities) which are treated as black-boxes exporting interfaces in line with object-oriented engineering principles.

A somewhat different but related idea of modularisation is discussed in [11], in which a modular BDI agent programming architecture is proposed, mainly targeted at supporting the design of specialised programming languages for single agent de-

velopment, and at providing transparent interfaces to existing mainstream programming languages for easy integration with external code and legacy software. The proposed architecture is independent to the internal structure of its components and agent reasoning model, and uses interaction rules to define the connections between the design components. This draws a clear distinction between knowledge representation issues and their dynamics, and promotes the design and development of specialized programming languages.

A goal-oriented approach to modularisation for cognitive agent programming languages is proposed in [17], suggesting agent goals as the basis of modularisation. The approach is then discussed providing a formal semantics in the context of the 3APL agent programming language. A similar notion has been proposed in the agent language GOAL [8] where a module is a component within an agent encapsulating policy-based intentions to be triggered in a particular situation. This approach combines the knowledge and skills to adequately pursue the goals of the agent in that situation and is used to realize a mechanism to control nondeterminism in agent execution.

A role-based approach to modularisation and reuse has been proposed in the context of *AgentFactory* agent platform and *ALPHA* programming language. To engender code reuse the framework makes use of the notion of commitments and role template [4].

Finally, to authors' knowledge the most recent approaches to modularity have been introduced in the 2APL and Jason agent platforms. In the former, similarly to the other related works, a module is considered as an encapsulation of cognitive components. The added value of authors' approach is the introduction of set of generic programming constructs that can be used by an agent programmer to perform a variety of operations on modules, giving agent programmers full control in determining how and when modules are used. In that way modules can be used to implement a variety of agent concepts such as agent role and agent profile [5]. The latter proposes a mechanism for modular construction of Jason agents from functionally encapsulated components – containing beliefs, goals and plans – so as to improve the support of the language

for the development of complex multi-agent systems, in an agent-oriented software engineering perspective [10].

In all these approaches modules are components *inside* agents. In this paper we explored a dual perspective, which allows for implementing modules as components *outside* the agents, externalised in proper tools and artifacts that agents can exploit (and possibly share) for their tasks. This allows for fruitfully integrated the approach described in this paper with existing ones, promoting a strong separation of concerns in programming agents, using – on the one side – agent language/architecture and related module mechanisms to define and modularise only those aspects that strictly concern agent internal aspects (state update and action selection in general, deliberation and means/ends reasoning in cognitive architectures); on the other side, artifact-based computational environments to engineer and modularise all those resources and tools that agents may exploit to achieve their tasks.

V. CONCLUSION AND FUTURE WORKS

In this paper we discussed a novel perspective to deal with agent modularisation in multi-agent system programming, based on the availability of artifact-based computational environments. The approach is not meant to be alternative to existing approaches, but rather a complimentary strategy which aims at improving the level of reusability, maintainability, extensibility – including dynamic extensibility – of multi-agent-based software systems.

Starting from this basic idea, now several points need to be further developed. The basic externalisation model must be improved so as to manage aspects related to protection: for instance, devising a strategy to prevent agents to access personal tools (modules) of other agents. Then, the language adopted to define the usage protocols, described in Section III, currently does not tackle some main problems that are important in the practice: two main ones are the management of name clashes (between the function and beliefs defined by the protocol and existing plans/goals/beliefs of the agents or other usage protocols) and the management of failures (currently mapped tout-court onto agent plan failure). Also no formal semantics has been devised yet. These points are part of future works. Also, the model currently adopted to describe usage protocols – in terms of function, preconditions, and a body – can be considered just a first step: some other further features will be explored, such as the possibility to define – besides preconditions – also *invariant* conditions, stating the conditions that must hold for all the duration of the usage protocol, and post-conditions, i.e. conditions that must hold when the protocol has completed. Besides conditions expressing the correctness of the protocol, tags could be used to support the reasoning about the tools (modules), such as an `effect` tag to specify the expected state of the artifact(s) and of agent beliefs by successfully executing the protocol, towards a truly cognitive use of artifacts/modules [13], [18].

Finally, in order to validate the approach, we plan to identify specific domains/applications to make it clear the advantages

of externalisation/internalization, eventually integrating different cognitive agent programming languages/platforms besides Jason, such as 2APL and Jadex.

REFERENCES

- [1] R. Bordini, J. Hübner, and M. Wooldridge. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons, Ltd, 2007.
- [2] L. Braubach, A. Pokahr, and W. Lamersdorf. Extending the capability concept for flexible BDI agent modularization. In *Programming Multi-Agent Systems*, volume 3862 of *LNAI*, pages 139–155. Springer, 2005.
- [3] P. Busetta, N. Howden, R. Ronquist, and A. Hodgson. Structuring BDI agents in functional clusters. In N. Jennings and Y. Lespérance, editors, *Intelligent Agents VI*, volume 1757 of *LNAI*, pages 277–289. Springer, 2000.
- [4] R. Collier, R. R. Ross, and G. M. O'Hare. Realising reusable agent behaviours with ALPHA. In *Multiagent System Technologies*, volume 3550 of *LNCS*, pages 210–215. Springer, 2005.
- [5] M. Dastani, C. Mol, and B. Steunebrink. Modularity in agent programming languages: An illustration in extended 2APL. In *Proceedings of the 11th Pacific Rim International Conference on Multi-Agent Systems (PRIMA 2008)*, volume 5357 of *LNCS*, pages 139–152. Springer, 2008.
- [6] D. Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.
- [7] D. Gelernter and N. Carriero. Coordination languages and their significance. *Commun. ACM*, 35(2):96, 1992.
- [8] K. Hindriks. Modules as policy-based intentions: Modular agent programming in GOAL. In *Programming Multi-Agent Systems*, volume 5357 of *LNCS*, pages 156–171. Springer, 2008.
- [9] R. Hubner, JomiFred Bordini and G. Picard. Jason and MOISE+: Organisational programming in the agent contest 2008. In *Dagstuhl Seminar on Programming Multi-Agent Systems*, volume 08361, 2008.
- [10] N. Madden and B. Logan. Modularity and compositionality in Jason. In *Proceedings of International Workshop Programming Multi-Agent Systems (ProMAS 2009)*. 2009.
- [11] P. Novák and J. Dix. Modular BDI architecture. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1009–1015, New York, NY, USA, 2006. ACM.
- [12] A. Omicini, A. Ricci, and M. Viroli. Artifacts in the A&A meta-model for multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 17 (3), Dec. 2008.
- [13] M. Piunti, A. Ricci, L. Braubach, and A. Pokahr. Goal-directed interactions in artifact-based MAS: Jadex Agents playing in CArtAgO environments. In *Proceedings of Intelligent Agent Technology 2008 (IAT '08)*. IEEE/ACM, 2008.
- [14] A. Ricci, M. Piunti, L. D. Acay, R. Bordini, J. Hubner, and M. Dastani. Integrating artifact-based environments with heterogeneous agent-programming platforms. In *Proceedings of 7th International Conference on Agents and Multi Agents Systems (AAMAS08)*, 2008.
- [15] A. Ricci, M. Piunti, M. Viroli, and A. Omicini. Environment programming in CArtAgO. In R. H. Bordini, M. Dastani, J. Dix, and A. El Fallah-Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications, Vol. 2*, pages 259–288. Springer, 2009.
- [16] A. Ricci, M. Viroli, and A. Omicini. The A&A programming model & technology for developing agent environments in MAS. In M. Dastani, A. El Fallah Seghrouchni, A. Ricci, and M. Winikoff, editors, *Programming Multi-Agent Systems*, volume 4908 of *LNAI*, pages 91–109. Springer, 2007.
- [17] M. B. van Riemsdijk, M. Dastani, J.-J. C. Meyer, and F. S. de Boer. Goal-oriented modularity in agent programming. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 1271–1278, New York, NY, USA, 2006. ACM.
- [18] M. Viroli, A. Ricci, and A. Omicini. Operating instructions for intelligent agent coordination. *The Knowledge Engineering Review*, 21(1):49–69, Mar. 2006.
- [19] D. Weyns, A. Omicini, and J. J. Odell. Environment as a first-class abstraction in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, Feb. 2007. Special Issue on Environments for Multi-agent Systems.

Call Graph Profiling for Multi Agent Systems

Dinh Doan Van Bien, David Lillis and Rem W. Collier
School of Computer Science and Informatics
University College Dublin
dinh@doanvanbien.com, {david.lillis, rem.collier}@ucd.ie

Abstract—The design, implementation and testing of Multi Agent Systems is typically a very complex task. While a number of specialist agent programming languages and toolkits have been created to aid in the development of such systems, the provision of associated development tools still lags behind those available for other programming paradigms. This includes tools such as debuggers and profilers to help analyse system behaviour, performance and efficiency. *AgentSpotter* is a profiling tool designed specifically to operate on the concepts of agent-oriented programming. This paper extends previous work on *AgentSpotter* by discussing its *Call Graph View*, which presents system performance information, with reference to the communication between the agents in the system. This is aimed at aiding developers in examining the effect that agent communication has on the processing requirements of the system.

I. INTRODUCTION

By its nature, a Multi Agent System (MAS) is a complex system consisting of loosely-coupled autonomous software entities that are required to communicate with one another in order to achieve individual or system objectives. To facilitate the development of such systems, a number of agent-oriented programming languages and MAS toolkits have been developed by a variety of researchers [1]. However, the availability of ancillary tools to aid with debugging and profiling is limited, particularly when compared with the available tools for other programming paradigms and languages.

Previous work introduced *AgentSpotter*, a profiling tool designed specifically for MASs [2]. Profiling is a performance analysis technique that is based on the notion that in a program, only a few places, called *bottlenecks* or *hot spots*, can account for the majority of the execution time of a program. Hence, by fixing only these sections of the code, the performance of a program can be substantially improved. Profiling was introduced almost 40 years ago by Donald E. Knuth in his empirical study of FORTRAN programs [3], and has since been successfully adapted to a variety of different languages, platforms and software architectures, including large distributed systems.

The aim of this paper is to continue the mapping of traditional concepts to those of agent oriented software engineering so as to facilitate the compilation of useful profiling data, presented in an intuitive, visual fashion in order to aid multi agent developers in improving the performance of their systems.

Section II provides a brief discussion of some related tools that have been developed for debugging and profiling MASs. In Section III, we give a brief overview of the *AgentSpotter* agent profiling application. Following this, in Section IV we

introduce the concept of a *call graph*, and analyse how the traditional concept of a call graph can be applied to a MAS. Section V presents the concrete implementation of an agent call graph within *AgentSpotter* profiling tool, followed by a discussion of the proposed approach in Section VII. Finally, we conclude and outline ideas for future work in Section VIII.

II. RELATED WORK

The work presented in this paper draws from two principal research areas. Firstly, in order to provide a profiling tool for MASs, it is necessary to examine the concepts and features of existing profiling tools for other programming paradigms, such as object-oriented programming. It is also necessary to explore the available programming tools aimed at aiding the debugging and profiling of MASs.

Initially proposed by Knuth, the key motivating factor behind profiling tools is his observation that “less than 4% of a program accounts for more than half of its running time” [3]. By identifying and improving code that represents a performance bottleneck, software developers can greatly improve the overall performance of their programs. An important motivator for the use of specialist profilers to identify these bottlenecks is the frequent tendency of developers’ mental map of their programming not matching the reality of how the program behaves. Thus, areas of concern that programmers may not have considered will be identified by the profiler.

In the context of more traditional, non-MAS, programming, developers generally have access to long-established and widely-accepted profiling tools such as *gprof* [4] or performance analysis APIs such as the Java Virtual Machine Tool Interface (JVMTI) [5] or *ATOM* [6]. However, those developing MASs do not tend to have access to such well-established tools.

One MAS framework that does provide the ability to glean data about system performance is *Cougar* [7]. This provides access to data on historical performance data, event detection, monitoring of ACL messages and a number of other services. The *LS/TS* agent platform provides an administrator tool that records some high-level system monitoring information [8]. The main limitation of these systems is the lack of post-processing of the raw performance data in order to produce meaningful synthetic indicators like a profiler would do.

Besides performance analysis, most agent frameworks provide a debugging tool similar to the *Agent Factory Debugger* [9], which provides information about the mental state and communication from the viewpoint of individual agents. A different type of debugging tool is the *Agent Viewer* that

is provided in the Brahms toolkit [10], which displays agent timelines so as to understand when agents' actions are taken.

As the work in this paper also requires the monitoring of inter-agent communication (see Section IV), it is also important to acknowledge the availability of existing communication-analysis tools for MAS platforms. A number of such tools have been developed for a variety of agent frameworks and toolkits to aid developers in understanding the interaction between agents in their systems. An early example of such a toolkit is Zeus [11], which contains a "society tool" that visualises the interaction between agents, so as to help in understanding the topology of the social contacts within the MAS. This type of tool also aids in debugging MASs, since developers can ensure that the expected communication and collaboration between agents is indeed taking place.

In the JADE agent development framework, a Sniffer Agent is a FIPA-compliant agent that monitors messages created with an Agent Communication Language (ACL) passed between agents and presents these in a simple graphical interface [12]. A more sophisticated tool, called *ACLAnalyser*, provides more detailed information on agent communication [13]. Again, the principal aim of this is to aid in debugging errors in MASs that relate to coordination or cooperation.

III. AGENTSPOTTER

AgentSpotter is a profiling tool designed specifically for gathering and displaying profiling information on MASs [2]. Figure 1 illustrates the abstract architecture of the system, designed to be compatible with any type of agent platform. The *AgentSpotter Service* runs within the Run-Time Environment of an Agent Platform, gathering data about the agents themselves (actions performed, messages exchanged), along with system data such as CPU and memory usage. This is the only portion of the system that must be ported in order to be run on different agent platforms. The data gathered is logged into a Snapshot File, which allows it to be accessed and analysed offline, once the system has finished running.

AgentSpotter Station is a visual application that provides a number of visualisations on various aspects of system performance, in order to help programmers to identify performance bottlenecks in their code.

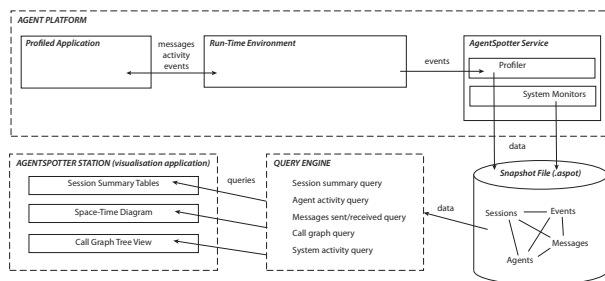


Fig. 1. AgentSpotter Architecture

The outputs utilised in this paper are gleaned from running a concrete implementation of the AgentSpotter service within the Agent Factory framework [14]. Agent Factory is a modular and extensible framework that provides comprehensive support for

the development and deployment of agent-oriented applications. A more detailed description of this implementation and the data gathered by AgentSpotter can be found in [2].

Previously, it was shown how AgentSpotter was used to map traditional profiling concepts onto agent-oriented concepts. This focused on two types of visualisation:

- **Flat Profile:** provides data on such things as agent activity, messages and reasoning/action duration in a tabular form.
- **Space-Time Diagram:** provides a navigable visualisation representing the data from the flat profile in a more intuitive manner.

The focus of this paper is on an agent-oriented call graph. Whereas a space-time diagram can aid in identifying the timing and extent of actions executed by agents, a call-graph is traditionally intended to also indicate the reasons why particular actions were undertaken at particular times.

IV. CALL GRAPH CONCEPT

A. Traditional Call Graph

The concept of a call graph was introduced in 1982 in the "gprof" profiling tool [4]. This is an improvement on the popular "prof" UNIX profiling tool. In addition to summarising the time spent in different functions, it also recursively presents all the call stacks annotated with the time spent in the various functions that are called. Another name for the call graph is "hierarchical profile", which conveys the idea that gprof provides information to aid in understanding the impact of one function in relation to all the functions that have called it.

Although the textual output of gprof is very dense and requires some practice to understand, modern profiler user interfaces have made call graphs more tractable by presenting them as tree view controls that can be interactively explored.

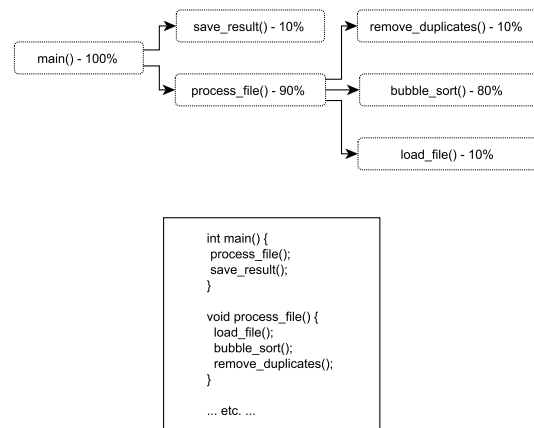


Fig. 2. Call Graph Tree View of a fictional C program that removes duplicate lines from a file

Figure 2 shows an example of a typical call graph. This relates to a fictional C language program that is designed to remove duplicate lines from a text file. In this tree view, the root node is the *main* function, which represents 100% of the total execution time of the program (including the execution time of its child nodes). Each node represents a function within

the program, with the child nodes representing functions that are called within the parent function. The percentages represent the cumulative proportion of the program's execution time that is attributable to a node and its children. In the example, the *main* function calls *process_file*, which is then represented as a sub-tree with leaves representing its own calls to the *bubble_sort*, *load_file* and *remove_duplicates* functions.

The key benefit of the call graph tree view is the extended context it gives to performance information. For instance, this simple example reveals that the program spends 90% of its time processing a file. The tree shows that one of the top-level function *process_file*'s callees, the *bubble_sort* operation, accounts for 80% of its caller time. A flat profile would have shown the time for these functions separately without explicitly showing the hierarchical link between them.

B. Agent-Oriented Call Graph Model

When constructing a flat profile for a MAS, it was necessary to map a number of concepts relating to traditional programming to equivalent concepts in the domain of agent-oriented programming [2]. A similar mapping must be performed in order to allow for the development of an agent-oriented call graph.

The central measure used in the traditional call graph is the function execution time. Each node represents a function, which can take the action of calling other functions as part of its execution. The consequence of this action is that some amount of time is spent executing the child function. Thus we can say that the *impact* of calling a function is that this additional processing time has been incurred.

In many MASs, agents tend to perform actions as a reaction to the receipt of ACL messages from other agents in the system. Thus in the same way the impact of a functional call in a traditional system is the execution time of that function, within a MAS, the impact of a message can be related to the additional processing that must be undertaken in order to react to the information contained therein, formulate a response or perform a requested task. Because of this mapping, we introduce, as a first simplified approach, the *agent message impact* measure to be used as an equivalent to the function processing time used in traditional profiling.

The quantification of such a measure is a difficult task, given the data typically available from MASs. One potential measurement for $T_{M_\alpha, B}$, the impact of a message M_α sent from an agent A to an agent B and received at time stamp α is to use the total amount of computation time used by the agent B until agent B receives a message M_Ω from another agent X at time stamp $\Omega \geq \alpha$. Let b be the duration of an activity by agent B at time stamp t where $\alpha \leq t \leq \Omega$. The impact of message M_α on agent B , $T_{M_\alpha, B}$, is then given by the recurrent equation:

$$T_{M_\alpha, B} = \sum_{t=\alpha}^{\Omega} b_t \quad (1)$$

In Figure 3 we have tried to summarise this concept in a graphical form. The diagram clearly shows that the three activity stars that lie between α and Ω make up the total impact

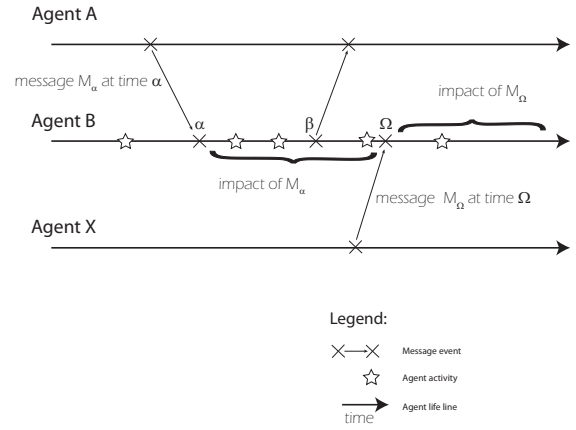


Fig. 3. Agent Message Impact Concept Diagram

of M_α on agent B . Note that the outgoing message at time stamp β does not break the computation sequence.

It is now easy to determine the total impact $T_{x,y}$ of all the messages sent by a given agent x to another agent y . Let M be the total number of messages sent, $1 \leq m \leq M$ a single message impact identifier, α_m the reception time stamp of message m from x to y , and Ω_m , where $\alpha_m \leq \Omega_m$, the next reception time stamp message coming right after m from any other source. The total impact $T_{x,y}$ is then given by the equation:

$$T_{x,y} = \sum_{m=1}^M \sum_{t=\alpha_m}^{\Omega_m} b_t \quad (2)$$

By applying the equations recursively, we can compute the total impact T_x of an agent x on N other agents numbered $1 \leq a \leq N$ as follows:

$$T_x = \sum_{a=1}^N \sum_{m=1}^{M_a} \sum_{t=\alpha_m}^{\Omega_m} b_t \quad (3)$$

Finally, the total impact T_S of all the K agents numbered $1 \leq k \leq K$ of a session S is given by the equation:

$$T_S = \sum_{k=1}^K \sum_{a=1}^{N_k} \sum_{m=1}^{M_a} \sum_{t=\alpha_m}^{\Omega_m} b_t \quad (4)$$

It must be noted that the total activity time A_S of the session S is given by the equation:

$$A_S = T_S + \sum_{k=1}^K \sum_{t=\alpha_S}^{\alpha_{k0}-1} b_t \quad (5)$$

where α_S is the first recorded time stamp in session S and α_{k0} the time stamp of the very first message received by agent k . To put it differently, the total impact for each agent can be computed only after it has received its first message.

This proposed method of calculating agent message impact is imperfect, and superior metrics are likely to be developed in the future. However, it does provide useful information for the debugging and development of MASs. Both the drawbacks

and benefits of this approach are outlined in more detail in Section VII.

V. CALL GRAPH VISUALISATION SPECIFICATION

The conceptual model we have presented deals with the session level, the emitter agent level, the receiver agent level and the message level. The graphical translation of the model, outlined In Figure 4, should be a tree view representing the levels we have previously enumerated plus an additional level for the FIPA ACL message content. A message content is defined as a performative plus an expression e.g. “request:doSomething(123)”. This additional level should give developers necessary contextual information for the messages. It is important to note that this fixed-depth call graph tree represents a divergence from traditional call graphs, whose depth is dictated by the depth of the deepest function call stack.

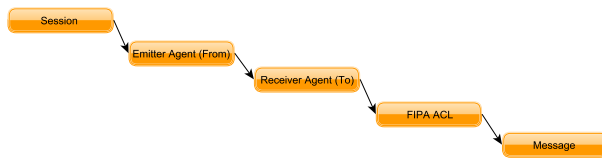


Fig. 4. Call Graph Tree View levels

The session at the root of the tree should add up to 100% of all emitter agents’ impact as defined by Equation 4. Then at each level, each node should recursively total the impact of its child nodes down to the message leaf nodes. These leaf nodes simply report their impact as defined by Equation 1. More precisely, at each level, for each node, the following values should be displayed:

- **Label:** informative text associated with the node. The structure of the label depends on the level as follows:
 - session: “capture date, time - duration”;
 - emitter agent: “from: agent id”;
 - sender agent: “to: agent id”;
 - FIPA ACL: “performative: contents”;
 - message: “sent: time stamp rec: time stamp”.
- **Total impact time:** sum of impact times of all the current node’s children.
- **% parent time:** percentage of the current node total impact time divided by the node’s parent total impact time.
- **% session time:** percentage of the current node total impact time divided by the session total impact time.

Ideally, developers should be able to order the intermediary tree levels differently so as to produce different call graph interpretations. For example, moving the FIPA level right above the emitter agent level would list for each FIPA ACL entry their total impact for all the emitter/receiver pairs.

A. User Interface

Despite having a fixed depth, a call graph tree view could potentially be very wide at the leaf level for sessions that produce thousands of messages. Therefore, to help developers

navigate easily through the tree, AgentSpotter Station offers an advanced tree navigation user interface that expands only that part of the tree which is currently explored so as to reduce the visual clutter. The currently explored part of the tree is highlighted in a different colour to give the developer some visual feedback.

Moreover, to speed up the retrieval of information on the system, a search feature allows developers to enter a keyword (e.g. an agent name or a performative). Doing so has the effect of highlighting in a special colour all the visible nodes that contain the specified keyword, significantly improving the visual retrieval speed of a node.

Finally, developers can zoom and pan around the tree view to locate items even more quickly.

B. Implementation

A sample screen shot of the visualisation of the call graph can be seen in Figure 5.

In this figure, the element numbered 1 on the screen shot is the tree root, i.e. the session level which represents 100% of the cumulative recorded activity time. The tree root is highlighted in blue because it is the current tree selection in this specific example. As such, it determines the branch that is expanded, as stated in Section V-A, so as to reduce the visual clutter. In order to provide a sufficient level of detail, all the children and grandchildren of a selected node are visible. Consequently, when the tree root is selected, only the first two subsequent levels are expanded, that is the emitter agent level and the receiver agent level. Hence, selecting an emitter agent node should make the FIPA ACL message level visible, and so on. As an illustration, the call graph numbered 3 shown in Figure 5 screen shot, has an agent receiver node selected; as a result, this branch is fully expanded down to the message impact level.

The element numbered 2 is a text area used to enter a search keyword. The number of nodes matching the keyword is displayed and all the matching nodes that are visible are highlighted in pink. For instance, the element numbered 3 is one of the nine nodes containing the “explorer” keyword and so is highlighted in the screenshot. In a large expanded tree, this highlighting greatly adds to the visual effect and consequently to the navigability of the tree. The bottom-most highlighted node in the tree represents a message sent from the “botagent3” agent to the agent named “explorer”. Clicking on this node would cause the subtree rooted at that node to be expanded so as to examine the content and timing of that message.

The visualisation is completely interactive and can be controlled using the mouse or the keyboard. Possible interactions include panning, scrolling, expand tree branches, zooming in and out.

One other important feature is the ability to alter the hierarchy of the nodes. Whereas the recursive nature of function calls means that these are inherently inflexible in the tree hierarchy they create, the nature of message-passing is a different situation. The hierarchy above places the sender of each message in a higher position in the hierarchy than the recipient. This means that the cumulative performance data

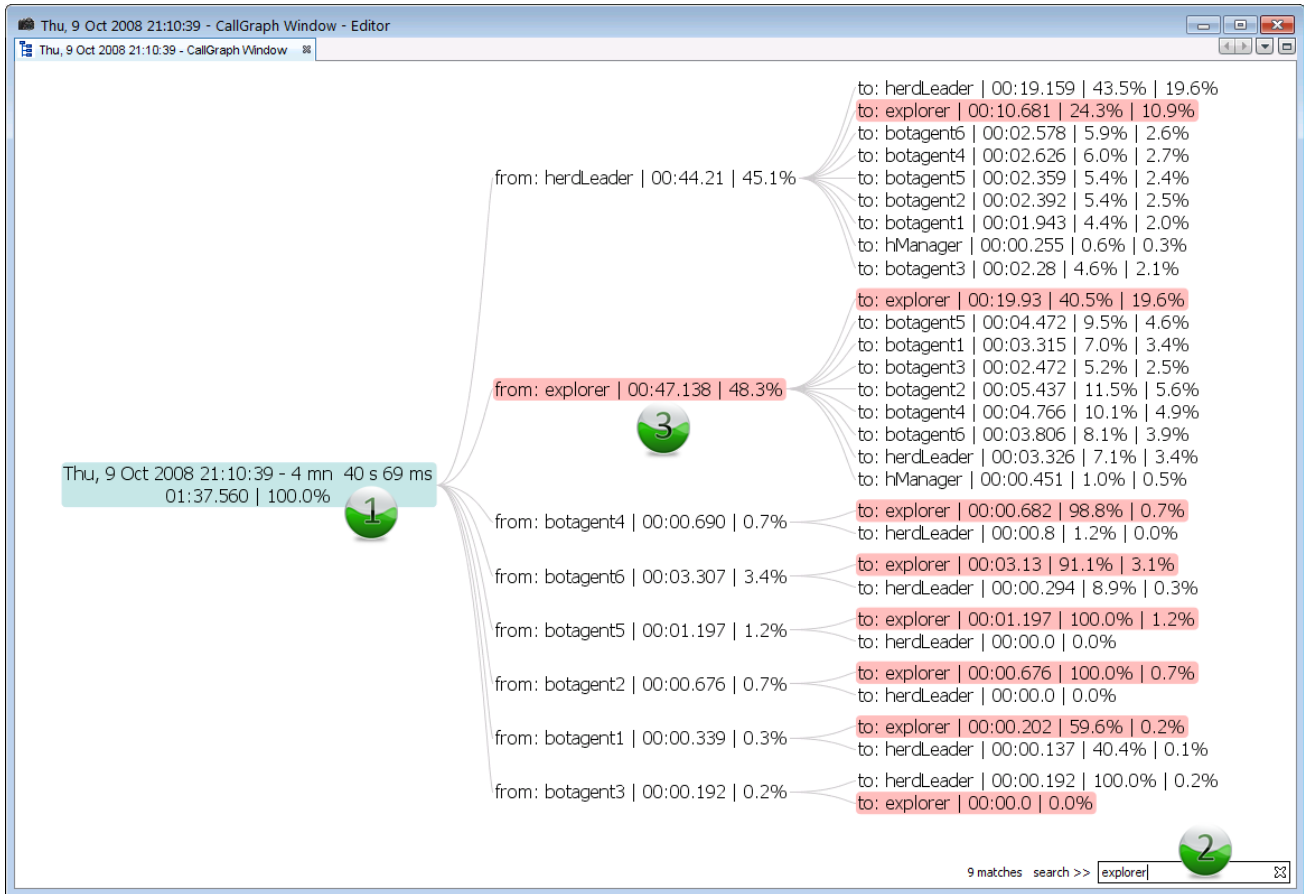


Fig. 5. Call Graph Tree View Screen Shot

for the higher-level nodes represents the overall impact of all messages sent by a particular agent to other agents. However, this may not encapsulate the information that a developer requires at a particular point in time. Changing the hierarchy to place the recipient agent above the sender changes the focus of the cumulative performance data. In this case, the figures represent the contribution to overall running time of a particular agent, based on the message that it receives from any and all sources. This may potentially identify entire individual agents as bottlenecks. This may be because the system's load is imbalanced, meaning that one agent may bear an inequitable share of the processing burden. Alternatively, in a distributed MAS, an agent may simply reside on a machine with inferior hardware resources. By exploiting the flexible nature of this hierarchy, users of the call graph tree view can alter the data being presented to better fit their needs.

VI. EVALUATION

To demonstrate the effectiveness of the call graph as a profiling tool, a simple benchmark application was developed. This consists of two types of agents. Overseer agents request worker agents to perform small, medium or large tasks. If a worker agent has recently been overloaded, it may refuse to execute the required task. Occasionally, overseer agents

will delegate the assignment of tasks to a worker agent, in which case the worker agent becomes an overseer agent for a brief period. A flat profile and space-time diagram for this benchmark system is contained in [2]. Figure 6 shows a portion of the call graph tree view for a run of this application. Here, the names of overseer agents begin with “master”, whereas the names of worker agents begin with “agent”.

The benchmark application profile (displayed in Figure 6 reveals that overseer agents master1 and master2 do not have the same impact on performance. Intuitively, one would expect each overseer agent to have an equal impact. However, in reality, we can see that the impact of messages sent by the “master2” agent accounts for only 20.4% of the overall session running time. Studying the call graph in more details helps in explaining this imbalance, by studying the effects of the messages with the content “pleaseDoThing(20)” that were sent by both master1 and master2 to agent001. These are emphasised in Figure 6 by means of the red rectangles. In each case, the parameter passed as part of the a “pleaseDoThing” request is related to the amount of work that the agent is being requested to perform.

The call graph shows that some requests from master2 have a 0.0 impact which in practice means they were ignored (no actions took place as a result of receiving those messages). In



Fig. 6. Benchmark Call Graph Tree View for master1 and master2

other words, when master1 sends a request to an agent, and immediately afterwards that master2 sends the same request to the agent, the overloaded agent simply refuses to execute the request. These “pleaseDoThing(20)” messages sent by master1 are reasonably consistent in terms of their impact, are never refused and account for a total of 8.8% of the total session running time. In contrast, only a single such request sent by master2 was honoured by agent001. This action accounted for a mere 0.5% of the session running time.

It is important to note that the greater impact of master1’s messages does not necessarily constitute a bottleneck, merely an imbalance in the system. This type of analysis would motivate the use of the space-time diagram to examine the timing of the messages in question, so as to further find why messages from master2 are more likely to be ignored by the worker agent.

A bottleneck would be identified by comparing the impact of different messages being sent (rather than the same message being sent by different agents). For instance, it is notable that the session impact percentages for “pleaseDoThing(1)” messages sent by master1 to agent001 are far lower than for “pleaseDoThing(20)”. In this simple benchmark application, this is an unsurprising result, as the increased workload is explained by the messages themselves, with the latter message requesting more processing to be undertaken by the former. However, figures such as these would indicate a bottleneck if the results are unexpected (i.e. where high-impact messages are not intended to trigger high-cost actions on the part of the message recipients) and so would motivate a closer examination of the longer-running actions to increase efficiency.

It may be possible to make such a deduction from viewing the underlying agent code itself, however the use of the call graph makes this far more easily apparent without the need for detailed examination of the code. This also means that testers that are not necessarily familiar with the code (or even perhaps testers who do not understand the programming language used) can identify bottlenecks and behavioural anomalies for developers to address.

VII. DISCUSSION

The proposed metric for measuring the agent message impact outlined in Section IV-B has a number of drawbacks. It operates on the naive assumption that the actions of an agent are directly related to the messages received by it. The impact of a message on an agent is thus taken as the sum of the execution times of all actions undertaken by the agent between the receipt of that message and the receipt of the next message.

The principal drawback with such an approach is that there is no provable causal link between the receipt of messages and the execution of actions. Agents may decide to act for reasons other than the receipt of ACL messages. For instance, a perceptor may have detected changes in the environment that may require some reaction. Also, when actions are executed as a direct consequence of the receipt of an ACL communication, there is no guarantee that all of the relevant actions have been performed prior to the receipt of the next message. Thus the agent message impact arising from the receipt of a single message may not be particularly informative.

Ideally, the best method of measuring the impact of the receipt of an ACL message would be to track the internal reasoning process of the agent, so as to identify those actions that are performed as a direct result of the receipt of a message and take only these into account when calculating the message impact. This is, however, a particularly difficult task, as the reasoning used by agents is extremely platform-dependent and would require a substantial amount of work to be performed in order to port AgentSpotter to other agent platforms and frameworks. This contravenes one of the fundamental aims of AgentSpotter, which is intended to be as platform-agnostic as is practicable.

Even if we are to settle for a framework-specific profiling system, the task of identifying direct causal links between events is also non-trivial. Whereas some agents may contain straightforward agent code that reacts to the receipt of a message by always invoking a particular action, this is unlikely to always be the case. Receipt of a message may alternatively lead to a refinement of an agent's goals, or even more subtly, an alteration of its current belief state, which in turn may result in goal refinements. Goals may be adopted based on the entire belief set, making it difficult to ascertain for certain whether the belief triggered by the message was a cause of the change in the agent's goals or a merely coincidental occurrence. Even when goals have been adopted, a plan selection algorithm is typically used to decide upon the best path to take towards satisfying those goals. Again, this is a potentially difficult process to trace reliably.

Although the proposed measure does have drawbacks and is somewhat simplistic, it is also important to highlight the benefits of such a measure. Whereas a high impact measurement for a single message may not be indicative of a major system bottleneck (and may indeed be merely coincidental), consistently high impact measures for similar types of messages are far more likely to be a result of a causal link between the receipt of the message and the processing that follows. It is this type of analysis that makes the call graph a useful tool in identifying situations that result in a high processing load and thus aid in helping developers concentrate on the appropriate portions of the code base to improve system efficiency.

VIII. CONCLUSIONS AND FUTURE WORK

We have proposed a new visualisation, the call graph tree view, in order to provide detailed information about the performance impact of agents interactions. After discussing the concept of call graph in a traditional programming context, we have then mapped it into an agent-oriented concept based on the idea that when an agent sends an ACL message to another, its impact on the amount of processing the recipient perform can be measured and used to identify system bottlenecks, load imbalances and efficiency issues. Although the proposed measure is not optimal, it does provide users with data that is appropriate and useful in the context of a profiler application. We have then extended this notion to a tree model with multiple levels: session, message emitter, message receiver, message. Finally, we have described the advanced user interface that allows developers and testers to interact with this model in the form of a zoomable and searchable tree view.

For further development of the call graph view of the AgentSpotter application, there are two principal areas for improvement. Firstly, as we have acknowledged in Section VII, the current measure for gauging the processing impact of a message being passed between agents is not an ideal one. We intend to investigate other possible measures that will include a stronger causal link between the receipt of a message and the resulting processing activity. In doing so, the other AgentSpotter views (flat profile and space-time diagram) will be utilised to ensure that any proposed measures reflect the reality of the system's execution as closely as possible.

The second significant area of future work is in the area of agent conversation protocols. The work presented in this paper considers each ACL message to be entirely independent of all other messages. The reality of agent communication is somewhat different. In the agent architecture presented as the benchmark application in Section VI, some overseer agents request that worker agents perform certain tasks. In our simple application, this is done by means of a single message containing the work request being sent to the worker. In reality, a more complex conversation would be used. The initial request for a task to be performed may be answered with an acceptance or rejection of the task being assigned, followed perhaps by the communication of the result of the task. Clearly, an agent accepting and performing a task will consume more processing resources than when it is rejected. However, in the existing model, both scenarios will be grouped together, under the initial message requesting action. Such behaviour may mask inefficiencies in the processing code by including the low-cost rejection actions in its session percentages. By introducing an additional conversation level into the tree, these situations can be separated, meaning that actions will be grouped according to entire agent transactions rather than single messages.

REFERENCES

- [1] R. Bordini, L. Braubach, M. Dastani, A. Seghrouchni, J. Gomez-Sanz, J. Leite, G. O'Hare, A. Pokahr, and A. Ricci, "A survey of programming languages and platforms for multi-agent systems," *Informatica*, vol. 30, no. 1, pp. 33–44, 2006.
- [2] D. Doan Van Bien, D. Lillis, and R. Collier, "Space-time diagram generation for profiling multi agent systems," in *Proceedings of PROMAS'09*, Budapest, Hungary, May 11-12th 2009.
- [3] D. E. Knuth, "An empirical study of FORTRAN programs," *j-SPE*, vol. 1, no. 2, pp. 105–133, Apr./Jun. 1971.
- [4] S. L. Graham, P. B. Kessler, and M. K. Mckusick, "Gprof: A call graph execution profiler," *SIGPLAN Not.*, vol. 17, no. 6, pp. 120–126, 1982.
- [5] Sun Microsystems, Inc., "JVM Tool Interface (JVMTI), Version 1.0," Web pages at <http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/> (accessed August 4th, 2008), 2004. [Online]. Available: <http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/>
- [6] A. Srivastava and A. Eustace, "Atom: a system for building customized program analysis tools," in *PLDI '94: Proceedings of the ACM SIGPLAN 1994 conference on Programming language design and implementation*. New York, NY, USA: ACM, 1994, pp. 196–205.
- [7] A. Helsing, M. Thome, T. Wright, B. Technol, and M. Cambridge, "Cougaa: a scalable, distributed multi-agent architecture," in *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, vol. 2, 2004.
- [8] R. Rimassa, M. Calisti, and M. E. Kernland, *Software Agent-Based Applications, Platforms and Development Kits*, ser. Whitstein Series in Software Agent Technologies and Autonomic Computing. Birkhäuser Basel, 2005, ch. Living Systems@Technology Suite, pp. 73–93.
- [9] R. Collier, "Debugging Agents in Agent Factory," *Lecture Notes in Computer Science*, vol. 4411, p. 229, 2007.

-
- [10] C. Seah, M. Sierhuis, W. Clancey, and M. Cognition, "Multi-agent modeling and simulation approach for design and analysis of MER mission operations," in *Proceedings of 2005 International conference on human-computer interface advances for modeling and simulation (SIMCHI'05)*, 2005, pp. 73–78.
 - [11] H. Nwana, D. Ndumu, and L. Lee, "ZEUS: An advanced tool-kit for engineering distributed multi-agent systems," *Applied AI*, vol. 13, no. 1, p. 2, 1998.
 - [12] F. Bellifemine, G. Caire, D. Greenwood, and E. Corporation, *Developing multi-agent systems with JADE*. Springer, 2007.
 - [13] J. Botia, J. Hernansaez, and F. Skarmeta, "Towards an Approach for Debugging MAS Through the Analysis of ACL Messages," *Computer Systems Science and Engineering*, vol. 20, 2005.
 - [14] R. Collier, G. O'Hare, T. Lowen, and C. Rooney, "Beyond Prototyping in the Factory of Agents," *Multi-Agent Systems and Application III: 3rd International Central and Eastern European Conference on Multi-Agent Systems, Ceemas 2003, Prague, Czech Republic, June 16-18, 2003: Proceedings*, 2003.

ReSeagent: A Refactoring Tool for Plan Level Refactoring in MAS Development

Ali Murat Tiryaki and Oguz Dikenelli
 Ege University, Department Of Computer Engineering
 35100 Bornova, Izmir, Turkey
 Email: {ali.murat.tiryaki,oguz.dikenelli}@ege.edu.tr

Abstract—The need for XP-like agile approaches that facilitate flexible evolutionary development has been widely acknowledged in the AOSE area. Such approaches improve acceptability of agent-technology by the industry. Evolutionary development of multi agent systems-MASs can only be applied successfully, if designs of the MASs being developed are improved throughout the development process. In our previous work, we have defined a refactoring approach that makes evolutionary MAS development possible. In this paper, we mainly aim to identify a development approach for MAS refactoring tools. In order to discuss this approach, we developed a refactoring tool called ReSeagent on the Seagent framework. Although the ReSeagent tool supports plan level refactoring patterns that are to be manually applied by the developers, ideas used in the implementation of this tool are generic ideas that provide a base for different refactoring types and development artifacts.

I. INTRODUCTION

Based on the experiences on agent-based system development, AOSE research community has realized that it is almost impossible to develop complex systems like multi agent systems - MAS in a sequential manner [33], [7]. The solution is the iterative approach which has been accepted as one of the best practices by software development community and integrated to all recent software development methodologies such as Rational Unified Process - RUP [26] and Extreme Programming - XP [3].

Managing the continuous evolution of software architecture and related design is one of the key issues in iterative development. XP introduces two critical practices to manage the evolution of software architectures: test driven development [21] and refactoring [18].

Test driven development produces test code for each class developed during iterations. This test code provides a protection shield against the flaws that can occur as a result of changes made on the working code by guaranteeing the functional accuracy of this code. The other best practice refactoring defines a process for improving the structure of the software system without altering the external behavior.

An iterative and incremental development life-cycle approach is quite appropriate for developing large scale distributed and complex systems such as MASs. XP-like agile processes, that introduce light-weight practices for iterative and incremental development in a controllable way, are needed to improve acceptability of the agent-technology by the industry [7], [33]. However, traditional testing and refactoring approaches and their supporting tools cannot be re-used directly

in MAS development, since MASs are built using different abstractions and techniques. So we need to re-define these practices for MAS development.

In [28], we have proposed a refactoring approach that makes evolutionary MAS development possible. This refactoring approach follows the route of traditional refactoring and provides some new refactoring patterns for MAS development. The proposed approach introduces some common problems called “bad smells” experienced during the development of MAS systems (such as duplicated behaviour structure and big plan) and maintenance strategies called “refactoring patterns” to overcome these bad smells. Each of the refactoring patterns defined in this proposed approach focuses on overcoming one or more than one bad smell(s) encountered during MAS development.

In this paper, we aim to introduce a basic development approach to produce MAS refactoring tools. Then, we implement a refactoring tool called as ReSeagent on Seagent MAS development framework [14], [15] by using the proposed approach. This tool supports refactoring on agent plans supported by almost all MAS development methodologies such as Passi [10], Tropos[5] and MaSe [13]. The tool can be used manually by developers during MAS development activities. ReSeagent focuses on refactoring Seagent plans whose meta model is defined clearly in Seagent. However, software architecture of the tool is generic and it can be used for other planning systems whose meta-models are explicitly defined or other MAS design artifacts such as role, goal and protocols.

II. RELATED WORKS

In the literature, there are some pioneering works which try to apply agile practices to MAS development.

Knublauch [20] used practices of extreme programming - XP [3], which is the one of the most known agile development processes used for MAS development. Although, this work proves the effectiveness of XP practices in terms of MAS development, refactoring is not explained in detail. Since the agent development framework and process meta-model, which are used during development, are very simple, refactoring operations on agents seem as very simple processes and refactoring practice is applied on agents. However, an agent that is developed by using a realistic development framework can play many roles in MAS and those roles have many goals, responsibilities and abilities. So, we believe that agents are not

small; on the contrary they are too big entities for testing and refactoring.

In another important work has been introduced by Chella et. al. [8], well known Passi methodology is transformed to Agile Passi. The testing framework developed by the Agile Passi research team provides an automated testing approach for testing multi-agent systems [6], [11]. Agile Passi approach does not introduce an iterative or evolutionary style for MAS development. Therefore, a refactoring approach that makes agile MAS development possible is not introduced in this work.

In [9], an agile methodology for MAS development is introduced. This methodology is a generic methodology based on the practices such as test driven development and refactoring that come from the agile approaches. The process of SADAAM consists of four key phases: design, test driven implementation, release & review and refactor & enhancement, that are applied iteratively until a finished state is reached. However, any detailed discussion on how the practices called test driven development and refactoring are applied during MAS development. Hence, the proposed methodology is a generic methodology that does not add too many specific ideas to the abstract development process proposed by agile approaches. To concretize how the proposed methodology is used for MAS development, the agile practices in the methodology have to be discussed exhaustively.

Another working [30] focuses on to define which and how traditional refactoring practice can be used for agent based simulation systems on a multi agent simulation systems modeling paradigm called MASim. At the end of the working, a catalog that consists of the refactoring patterns used to improve the system designs based MASim is introduced. Moreover, some of the proposed refactoring patterns has been implemented and integrated to an agent based simulation platform called SeSam. This working does not introduce a general refactoring approach and its main characteristics for MAS development. The proposed refactoring patterns can be used to only the agent systems that have a specific type. To define the refactoring patterns that can be used during MAS development, firstly a general refactoring approach based on the characteristics of these systems has to be defined.

In [29], an iterative and incremental development approach called agent oriented test driven development - AOTDD is proposed to handle the complexity and continuously changing nature of the requirements in MAS development. In AOTDD, developers follow the development cycle with adding the new functionalities to the system between iterations, just like all other agile & iterative development approaches. Also, the life cycle of proposed test driven approach and a testing tool that supports the proposed test driven approach are introduced in this work. Since this work is focused on the testing part of test driven development, the refactoring step that is very critical for iterative and incremental development is not discussed in detail.

These works neither propose a systematic approach for MAS refactoring nor introduce a refactoring tool support for these approaches. In this paper, we focus on these two points.

III. A DEVELOPMENT APPROACH FOR MAS REFACTORIZING TOLLS

Refactoring is directly dependent on the executable artifacts of the developed system. In AOSE area, it is not possible to define a set of executable artifacts that can be agreed on, since there are several agent architectures such as BDI, re-active, self-organized and several development approaches such as Gaia [32], Tropos [5] and Adelfe [4] that aim to develop agents based on these agent architectures. Naturely, different executable artifacts may emerged based on the used agent architecture and/or development frameworks that support these architectures. As a conclusion, it is impossible to develop a refactoring tool that is usable for all kinds of MAS development artifacts. Instead of this, we need a generic approach to develop MAS refactoring tools. This approach can be specified for different MAS architectures and/or different artifacts to produce a suitable refactoring tool.

The proposed development approach has three steps listed as follows:

- 1) Define the meta-model of the target executable development artifacts,
- 2) Define the bad smells encountered during the development of the artifacts specified in the previous step,
- 3) Define the refactoring patterns that overcome the bad smells defined for the target development artifact.

To illustrate how the proposed approach can be applied to refactoring tool development, we chose the agent plans as the target development artifact. Agent plan abstraction is one of the most common artifacts in MAS development methodologies. Almost all of the MAS methodologies use plan abstraction (it is named with different terms in different methodologies) to model agents' internal behaviours. For example, in the goal oriented development approach, each agent goal is achieved by one or more agent plan(s). Several approaches can be used to build plan structures such as HTN [31], [23] based on the agent infrastructure. Developers have to specify the planning approach before they apply first step of the proposed refactoring tool development approach (defining the meta-model of the target artifacts).

In our case, we aim to develop a refactoring tool for the Seagent framework that was developed by our research group. So, our refactoring tool called ReSeagent aims to refactor agent plans that are directly dependent on the Seagent planning infrastructure.

The rest of this section explains how the steps of the proposed approach were applied for the plan artifact in the Seagent framework.

A. Define the meta-model of the target executable development artifacts

Performing a refactoring pattern requires a clear understanding of the abstract syntax and semantics of both the source and target models. Meta-modeling is a common technique for defining the abstract syntax of models and the interrelationships between model elements. To identify refactoring patterns and bad smells for an executable design artifact, we have to

know the meta-model used to build structures of this type of artifacts.

In Seagent framework, the hierarchical task network - HTN formalism [23], [31] is used to build and store of agent plans. Hence, the plan level refactoring patterns introduced in this paper are dependent on the HTN formalism. ReSeagent refactoring tool supports to apply the mechanics of these HTN dependent refactoring patterns on HTN plans that are developed through Seagent. The structure of the HTN meta model used in the current Seagent version is shown in the figure 1.

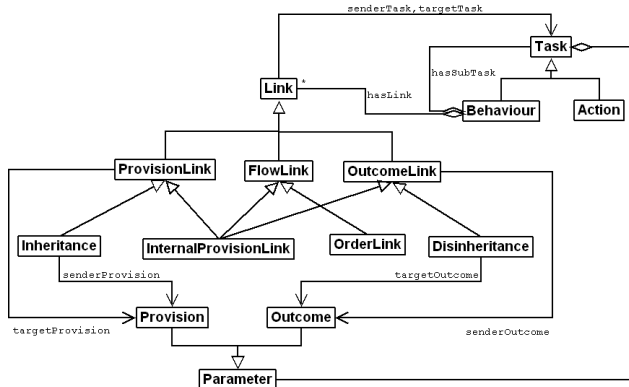


Figure 1. Seagent HTN Meta Model

In HTN formalism, there are two kinds of task; complex task (we call behaviour) and primitive task (we call action). Complex tasks hold the structure of its sub-tasks and links between these tasks. Primitive tasks have directly executable code. Information requirements of tasks are illustrated as provisions. Outcomes are result states of tasks. Data is transferred to other tasks through the several kinds of link. An inheritance link is used to transfer a provision of a parent complex task to a sub task. Disinheritance links are used to transfer outcomes of sub-tasks to parent complex task. And finally, the information flow between outcomes and provisions of the tasks in the same level is provided by internal provision links. As an addition of the links come from the HTN formalism, there is one more type of link called order link in the Seagent behavioural PSM. Order links do not pass any information between tasks. They are used only to order the execution of tasks.

B. Define the bad smells encountered during the development of the artifacts specified in the previous step,

To make decision about applying refactoring patterns on an artifact, we need to define the common design problems that are encountered by developers during the system development process frequently. Fowler named these common problems as bed smells in [18] and introduced refactoring patterns to overcome these bad smells in software design. Each of the defined refactoring patterns is introduced to overcome one or many bad smell(s).

Similarly, we have to define bad smells for MAS development. Based on our experiences on the MAS development, we have identified following bad smells for plan level refactoring: Duplicated behaviour structure, execution decision in a plan, big behaviour, incoherent behaviour, redundant task group.

Detailed definitions of all defined bad smells are accessible through the Seagent web side¹.

C. Define the refactoring patterns that overcome the defined bad smells for the target development artifact.

To develop a refactoring tool that can be used during MAS development, the refactoring patterns that define the order of the mechanics for each refactoring pattern has to be defined by a standard way. Each pattern includes pre-defined mechanics in order to overcome one or more bad smell(s). These mechanics has to be defined by using a machine understandable semantic. The tool uses the pre-defined machine understandable pattern definitions to overcome bad smells.

In ReSeagent, we have defined seven refactoring patterns for HTN based plan refactoring. These plan level refactoring patterns that were defined based on our experiences with its initiator bad smell(s) are shown in table I.

Refactoring	Bad Smell
Extract Plan	Execution Decision in Plan
Extract Behaviour	Big Behaviour Duplicated Behaviour Structure
Behaviour to Plan	Execution Decision in Plan
Rename Behaviour	Incoherent Behaviour
Replace Tasks with a Task	Redundant Task Group
Deliver from Behaviour	Incoherent Behaviour
Extract Super-Behaviour	Duplicated Behaviour Structure

Table I
REFACTURING PATTERNS AND THEIR INITIATOR BAD SMELLS IN THE PLAN LEVEL

The detailed definitions of all refactoring patterns defined are accessible on the refactoring patterns part of the Seagent web side.

IV. A HYBRID APPROACH FOR DEFINITION OF REFACTURING PATTERNS

To implement a refactoring tool, firstly the refactoring techniques supported by the tool have to be defined in a usable form for the tool. We used a hybrid approach based on the mechanisms used for defining transformation rules in the Model Driven Engineering - MDE to define the refactoring techniques mentioned in the previous section for our refactoring tool. MDE is a discipline in software engineering that relies on models as first class entities and that aims to develop, maintain and evolve software by performing model transformations [22], [25]. This section introduces our hybrid approach that is based on the model transformation techniques [19] and used to define refactoring techniques in ReSeagent refactoring tool.

¹http://etmen.ege.edu.tr/wiki/index.php/refactoring_agent_system

From the MDE perspective, a refactoring operation is a model transformation between the initial and the improved models. Model is a structure of design artifacts in the refactoring context. Like each model transformation has some transformation rules, each refactoring has some mechanics that achieve main goal of the refactoring by working together. It is nature to think that the refactoring approaches and tools can be based on the model transformation approaches and techniques. To develop a refactoring tool by using the model transformation techniques, mechanics of refactorings should be defined as the transformation rules. In this manner, the refactoring tool operates the refactoring mechanics that are defined as transformation rules according to the refactoring pattern in order. This is not a new idea and there are some works about rule based refactoring in the literature such as [2], [24]. All of these works aim to define refactoring operations by rules for their own target development style.

Transformation rules are the focal point for model transformation. The techniques and languages that are used to identify these rules specify the main characteristics of model transformation. There are two main approaches to define transformation rules; declarative and imperative approaches [12]. Declarative approaches (e.g., [1]) are attractive because particular services such as source model traversal, traceability management and automatic bidirectionality can be offered by an underlying reasoning engine. On the other hand, imperative approaches (e.g., [27]) may be required to implement transformations for which declarative approaches fail to guarantee their services. Especially when the application order of a set of transformations needs to be controlled explicitly, an imperative approach is more appropriate thanks to its built-in notions of sequence, selection and iteration.

In Seagent framework, plan models are stored in ontologies that are built by using a description logic based language called Web Ontology Language - OWL². Refactoring operations on these plan models can be considered as model transformations between initial plan ontologies and improved plan ontologies. So, a logic based declarative approach looks like appropriate for building rules on the these models. A refactoring tool that supports transformation between OWL ontologies using logic based rules is useful for refactoring Seagent plan models. However, logic based rules are not enough for defining refactoring mechanics because of the two handicaps of logic based declarative approaches listed below:

1- Almost all of the logic languages such as Prolog have been developed to extend their target models. By using these languages, new definitions can be made on the existent elements in the model. These languages do not support to remove and change the existent elements in the model. Refactoring techniques require removing and changing of the existent elements in the model besides extending of the model.

2- Another handicap of the declarative approach for creating transformation rules of refactorings is that the mechanics of refactorings should be operate in sequence, As mentioned above, such a sequence operation can be controlled explicitly by an imperative approach.

Because of these handicaps, the declarative approach cannot be used alone to define refactoring mechanics since their handicap mentioned above. To define refactoring mechanics as rules, imperative or hybrid approaches should be used.

ReSeagent uses a hybrid approach that combines the advantages of declarative and imperative rule definition approaches. The declarative side of this approach is achieved by using Semantic Web Rule Language - SWRL³. SWRL is a logic based rule language that supports defining rules on OWL ontology models. Refactoring mechanics that extend the initial models are defined as SWRL rules. Refactoring mechanics defined using the HTN paradigm forms the imperative side of our hybrid approach[31], [23].

In this hybrid approach, each refactoring pattern is defined as an agent plan by using the HTN paradigm and each refactoring mechanic is implemented as an executable action in refactoring plans. Refactoring mechanics that extend the initial models are implemented as rule actions. Rule actions have the responsibility of operating a SWRL rule on the OWL model of the related Seagent plan. Refactoring mechanics that change and/or remove the existing element in the initial model are implemented as normal actions. Normal actions implement change and removal activities by means of Java code that handles the ontology explicitly. All of the actions that are implemented to realize the mechanics of a refactoring pattern compose a refactoring plan that achieves the main goal of the refactoring pattern. Sequential execution of these actions is controlled by the plan structure.

Many different methods such as finite state machine can be chosen to implement the imperative side of our hybrid rule definition approach. However, defining each refactoring pattern as an agent plan by using the HTN paradigm in ReSeagent has some advantages listed below:

- **Simplicity:** In Seagent, HTN paradigm is used for testing and implementation of plans. Defining refactoring patterns by using the same method simplifies the addition of new refactoring patterns into the refactoring tool.
- **Reusability:** HTN paradigm makes it possible to reuse other pre-defined plan structures in higher level plan structures. Thanks to this, big refactorings can be simply implemented by re-using the pre-defined refactoring plans in a high level refactoring plan.
- **Generality:** Since the refactoring plans are agent plans like domain dependent user plans, these refactoring plans can also be refactored by applying refactoring plan(s) on these plans.

The software architecture of ReSeagent that executes the refactoring patterns defined by our hybrid approach is explained in the following section.

V. OVERAL SOFTWARE ARCHITECTURE OF RESEAGENT

ReSeagent refactoring tool was implemented as a plug-in on the Seagent plan editor in the Seagent Development Environment - SDE like the refactoring support of Eclipse. ReSeagent gives support for refactoring the plan models developed using

²<http://www.w3.org/TR/owl-features/>

³<http://www.w3.org/Submission/SWRL/>

SDE. The tool applies pre-defined refactoring patterns on the related plan models to fulfill the refactoring requests received from Seagent plan editor.

ReSeagent focuses on the refactoring of Seagent plans whose meta-model is clearly defined in Seagent. Additionally, since the software architecture of the tool is generic and it can be used for other planning systems whose meta-models are different or for other MAS design artifacts such as role, goal and protocols. To add support for the artifacts that have different meta-models, you have to add new refactoring plans that work on these meta-models into the refactorer role of ReSeagent, and make some additions to the initiator module for initiating these new refactoring plans.

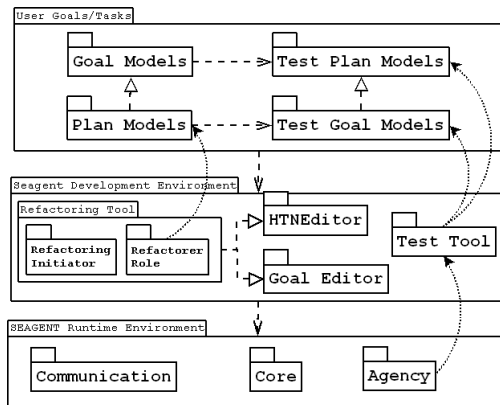


Figure 2. Overall Software Architecture of ReSeagent

SDE has the responsibility of developing executable artifacts such as plan models and goal models that can be executed by Seagent. For this purpose, SDE includes a plan and a goal editor for developing plan and goal models. It includes also a testing tool called SeaUnit that verifies the functionalities of these development artifacts [16]. The overall software architecture of ReSeagent and its dependencies on other modules in SDE is shown in figure 2.

ReSeagent consist of two sub-packages: refactoring initiator and refactorer role. Refactoring initiator package was developed as a plug-in on the Seagent Plan Editor like the refactoring support of the well known Eclipse environment. It has the responsibilities of capturing the refactoring requests in the plan editor, and initiating a refactoring operation for each of these refactoring requests. On the other hand, refactoring role holds the refactoring plans that are defined to provide the mechanics of the refactorings.

A. Refactoring Initiator

Refactoring initiator module of ReSeagent has a simple structure. This module has the responsibilities of capturing the refactoring requests from the plan editor and the user preferences that are needed to fulfill these requests, and initiating a refactoring operation that is suitable for each of the captured refactoring requests.

When the developer wants to start a refactoring operation on a pre-defined plan model by using the refactoring plug-in of the Seagent plan editor, the initiator initiates an agent that plays the refactorer role and then passes the refactoring request with its inputs to this refactorer agent. This agent maps the received refactoring request to a suitable refactoring plan situated in the plan library of the refactorer role, passes the input values to this plan and executes it. During the plan execution, the input values of the refactoring requests are passed to the sub-tasks of the plan via the links in the HTN structure. Then, these subtasks are executed depending on their order in the HTN structure. At the end of the plan, updated plan models are returned to the initiator via the outcomes so that the results can be shown to the plan editor user.

There can be many plans that realize the same refactoring operation by means of different ways. In such a situation, refactorer agent decides which refactoring plan has to be executed, according to inputs of the refactoring request and its internal state. This ability of the refactorer agent comes from Goal Mapping Engine in Seagent framework [17]. Thanks to this engine, Seagent agents can map a request to most suitable of many plans that achieve same goal in different ways. When this decision is made, some criteria such as inputs and outputs of the goal are considered by the Goal Mapping Engine. To add more than one refactoring plan that can realize the same refactoring operation to Seagent, it is enough to add correct mapping definitions to the knowledge base that are used for goal mapping (called Goal Mapping Ontology in Seagent) by the refactorer agent .

Another responsibility of the initiator is passing the updated plan model(s) to the plan editor. Each refactoring plan has some outcomes that return the plan models that have been changed during the execution of the refactoring plan. There is one outcome for each updated plan model. Initiator listens to the planner of the refactorer agent after it initiates this agent. When a PlanFinished event is thrown by the planner, the initiator captures the updated plan models returned by the outcomes and updates the model(s) in the plan editor. Hereby, new structures of the refactored plans are shown to the user.

B. The Refactorer Role

Refactorer role is a special role that has refactoring goals and refactoring plans that achieve these goals. This role can access plan models and action definitions in the system. This role has the responsibility of applying plan level refactorings on plan models during MAS development.

The refactorer role has refactoring goals that aim to apply a specific refactoring pattern on the Seagent plan models. Each of these refactoring goals is achieved by one or more than one refactoring plans. So, the plan library of this role has to include at least one refactoring plan for each refactoring pattern supported by ReSeagent tool.

The current version of ReSeagent refactoring tool supports the following plan level refactoring patterns: *Replace Tasks with a Task*, *Extract Behaviour*, *Behaviour to Plan*, *Extract Plan*.

The detailed definitions of these refactoring plans can be found on the ReSeagent plans page of the Seagent web site.

Also, the OWL ontologies of the plans and Java codes of the actions in these plans are downloadable on this page.

To give an inside about the refactoring plans in ReSeagent, one of the refactoring patterns and the refactoring plan developed to achieve this refactoring in ReSeagent are explained in the following section.

Replace Tasks with a Task Refactoring and Its Implementation in ReSeagent

Replace Tasks with a Task refactoring can be used in such a case: a functionality achieved by more than one tasks in a plan structure can be achieved by only one task. This refactoring removes these tasks from the plan structure and adds the new task to the plan structure instead of the removed tasks.

When the *Redundant Task Group* bad smell is realized in a plan structure, the plan can be made more readable and simpler by replacing the task group with a task that can achieve same functionality.

To apply *Replace Tasks with a Task* refactoring, the new task has to have all of the provisions and outcomes of the tasks in the replaced task group that are linked to other tasks in the plan structure.

Mechanics:

- 1) Remove the task group from the plan and add the new task.
- 2) Find inheritance and provision-outcome links that are attached to the provisions of the replaced tasks and attach such links to the suitable provisions of the new task.
- 3) Find disinheritance and provision-outcome links that are attached to the outcomes of the replaced tasks and attach such links to the suitable outcomes of the new task.
- 4) Scan the provision-outcome links whose source or target task(s) is the reference to the replaced tasks. If there are such links, remove these links from the plan structure.

Replace Tasks with a Task refactoring plan in ReSeagent refactoring tool is developed to realize the goal of “replacing a task group with a task that can fulfill same functionality” which is the aim of *Replace Tasks with a Task* refactoring pattern. HTN structure of this refactoring plan that implements the mechanics of *Replace Tasks with a Task* refactoring pattern is shown in figure 3.

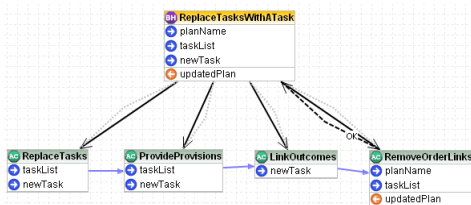


Figure 3. HTN structure of the *Replace Tasks with a Task* refactoring plan in ReSeagent

The plan takes the name of the plan whose structure is be refactored, the list of the tasks that are to be removed from the plan structure and the task that is to be added into the plan structure as provisions, and transfers these provisions

to its sub-tasks. The plan has four actions. Each of these actions achieves one of the mechanics of the *Replace Tasks with a Task* refactoring pattern. These actions which are called *ProvideProvisions* and *LinkOutcomes* extend the model of the target plan structure by adding new links. Hence, each of these actions operates a SWRL rule to fulfill its responsibility. At the end of the plan execution, updated plan structure is returned through an outcome.

VI. CASE STUDY

In this section, we introduce a case study that shows the usage of our refactoring approach and ReSeagent refactoring tool during the development of an actual MAS application which is a conference management system that has been developed by Seagent group.

At the beginning of the development, we did not intervene the developers and let them to follow a sequential development process that does not impose evolutionary development. The developers developed some of the goals such as “building the program committee”, “sending call for paper” and “initiating a conference” by applying the activities of their development process.

After developing a few of the system goals, some bad smells emerged in the design of the MAS that was developed: some plan structures were duplicated in many plans. Furthermore, the developers were disappointed from the unmanageable structures of the plans developed.

We can give an example for these bad smells on a simple plan structure from the conference management MAS. This plan structure achieves the “sending call for paper” goal of the “organization” role in this system. The initial HTN structure of this plan that was obtained at the end of the sequential development process for “sending call for paper” goal is shown in figure 4.

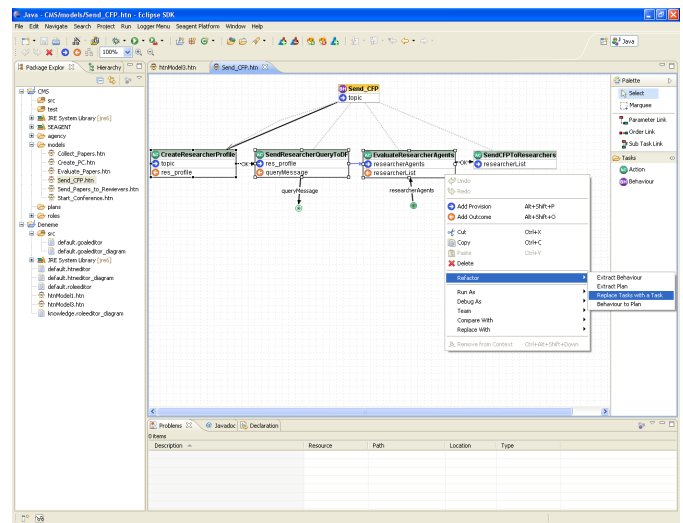


Figure 4. The initial plan structure of the Send_CFP plan

The simple plan in the figure 4 takes the conference topic as a provision. This provision is passed to the “create suitable researcher profile” action through an inheritance link. In this action, a researcher profile object is created, the interested_topic

field of this profile is set with the topic that is received as a provision and this researcher profile is returned through the "OK" outcome. The other action called "prepare and send query message to DF" takes the researcher profile, creates a query message by using this profile and sends this message to directory facilitator - DF. The "evaluate incoming researchers" action has an external provision called researcherList. This provision includes agent descriptions of the researcher agents that are sent by the DF. In this action, the description of the researcher agents are filtered according to the preferences and suitable researchers are selected. The final action called "send CFP to selected researchers" has the responsibility of sending call for paper of the conference to selected researchers using the agent descriptions that are received as a provision.

Some tasks in the plan structure have a common goal called "finding the suitable researcher agents" that should be tested independently from the other goals. This goal can also be part of the other plan structures such as "create program committee" in the system. This was a bad smell called *Duplicated Behaviour Structure*. So, we decided to collect the actions called *CreateResearcherProfile*, *SendResearcherQueryToDF* and *EvaluateResearcherAgents* into a new plan that achieves the common goal by applying *Extract Behaviour* refactoring on these actions.

To initiate an *Extract Behaviour* refactoring operation on these three actions, we selected these actions and right clicked the mouse. In the opened menu, we chose the Refactor —> Extract Behaviour as shown in figure 4. Then, a window related with the *Extract Behaviour* refactoring appeared on the screen. This window had three spaces that had to be set by the input values of the plan. These values were plan name, tasks list and new task. When we set all of the spaces by values and then clicked to the start button, the refactoring operation was initiated. After, a short duration, the improved structure of the plan appeared in the Seagent plan editor.

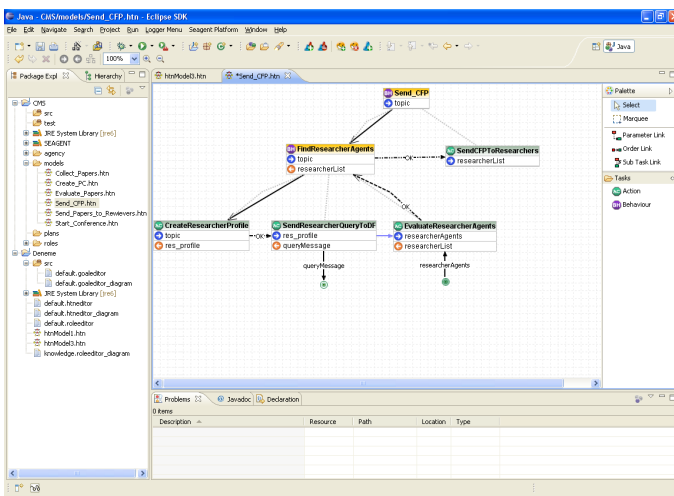


Figure 5. Final structure of the Send_CFP plan

At the end of the *Extract Behaviour* refactoring, we obtained a new plan called *FindResearcherAgents* that can be re-used in the other plan structures. This plan has the responsibility of finding agent descriptions of the researchers that work on

the conference topic according to the topic provision. The *FindResearcherAgents* plan was simply used in some other plans in conference management system. The plan structure of our "send CFP" plan after the *Extract Behaviour* refactoring is shown in figure 5.

After the refactoring operations on the developed plan structures, we obtained more readable and manageable plans for the conference management system. Reusable plan structures such as *FindResearcherAgents* obtained during the refactoring operations simplified to development of other system goals that include the goals achieved by these plans.

VII. CONCLUSION

In this paper, we define a development approach for refactoring tools that can be used during the development of MASs. To discuss this development approach on a refactoring tool implementation, a refactoring tool called ReSeagent has been implemented on the Seagent framework by following the process proposed by this approach. This tool supports the manual application of pre-defined refactoring patterns stored as agent plans on Seagent plan models during the development activities. So far, ReSeagent refactoring tool was used in the development of the several MAS applications such as e-barter and conference management systems developed by the Seagent group. The experiences we obtained during the development of these systems show that ReSeagent refactoring tool facilitates evolutionary MAS development by simplifying the refactoring process.

REFERENCES

- [1] David H. Akehurst and Stuart J. H. Kent. A Relational Approach to Defining Transformations in a Metamodel. In *The Unified Modeling Language: Model Engineering, Concepts, and Tools*, volume 2460 of *Lecture notes in computer science*. Springer, 2002.
- [2] Joachim Baumeister and Dietmar Seipel. Verification and refactoring of ontologies with rules. In Steffen Staab and Vojtech Svatek, editors, *EKAW*, volume 4248 of *Lecture Notes in Computer Science*, pages 82–95. Springer, 2006.
- [3] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.
- [4] Carole Bernon, Valérie Camps, Marie P. Gleizes, and Gauthier Picard. Engineering Adaptive Multi-Agent Systems: The ADELFE Methodology. In Brian H. Sellers and Paolo Giorgini, editors, *Agent-Oriented Methodologies*, pages 172–202. Idea Group Pub, NY, USA, juin.
- [5] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.
- [6] G. Caire, M. Cossentino, A. Negri, A. Poggi, and P. Turci. Multi-agent systems implementation and testing. In *From Agent Theory to Agent Implementation, Fourth International Symposium (AT2AI-4)*, 2004.
- [7] L. Cernuzzi, M. Cossentino, and F. Zambonell. Process models for agent-based development. *Journal of Engineering Applications of Artificial Intelligence*, 18 (2), 2005.
- [8] A. Chella, M. Cossentino, L. Sabatucci, and V. Seidita. From pass to agile pass: Tailoring a design process to meet new needs. In *IEEE/WIC/ACM International Joint Conference on Intelligent Agent Technology (IAT-04)*, 2004.
- [9] Neil Clynch and Rem Collier. Sadaam: Software agent development an agile methodology. In *Proceedings of the Workshop of Languages, methodologies, and Development tools for multi-agent systems*, 2007.
- [10] Massimo Cossentino, Luca Sabatucci, and Antonio Chella. Patterns reuse in the pass methodology. In *Proceedings of the Fourth International Workshop Engineering Societies in the Agents World (ESAW'03)*, pages 29–31. Springer-Verlag, 2003.

- [11] Massimo Cossentino and Valeria Seidita. Composition of a new process to meet agile needs using method engineering. In *SELMAS*, pages 36–51, 2004.
- [12] Krzysztof Czarnecki and Simon Helsen. Classification of model transformation approaches. In *OOPSLA-03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, 2003.
- [13] DeLoach S. A. Multiagent Systems Engineering A Methodology and Language for Designing Agent Systems. In *Proc. of Agent Oriented Information Systems*, pages 45–57, 1999.
- [14] Oguz Dikenelli, R. C. Erdur, O. Gumus, E. E. Ekinci, O. Gurcan, G. Kardas, Inanc Seylan, and Ali Murat Tiryaki. Seagent: a platform for developing semantic web based multi agent systems. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1271–1272, New York, NY, USA, 2005. ACM Press.
- [15] Oguz Dikenelli, Riza Cenk Erdur, Geylani Kardas, Ozgr Gumus, Inanc Seylan, Onder Gurcan, Ali Murat Tiryaki, and Erdem Eser Ekinci. Developing multi agent systems on semantic web environment using seagent platform. In *Engineering Societies in the Agents World VI*, volume 3963 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2006.
- [16] Erdem Eser Ekinci, Ali Murat Tiryaki, and Oguz Dikenelli. Goal oriented agent testing revisited. In *Agent Oriented Software Engineering 2008*. Springer Verlag, 2008.
- [17] Erdem Eser Ekinci, Ali Murat Tiryaki, Onder Gurcan, and Oguz Dikenelli. A planner infrastructure for semantic web enabled agents. In *OTM Workshops*, volume 4805 of *Lecture Notes in Computer Science*, pages 95–104, Vilamoura, Algarve, Portugal, 2007. Springer.
- [18] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Boston, MA, USA, 1999.
- [19] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained: The Model Driven Architecture—Practice and Promise*. Addison-Wesley Professional, April 2003.
- [20] Holger Knublauch. Extreme programming of multi-agent systems. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 704–711, New York, NY, USA, 2002. ACM Press.
- [21] Johannes Link and Peter Frolich. *Unit Testing in Java: How Tests Drive the Code*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.
- [22] Tom Mens and Pieter Van Gorp. A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142, March 2006.
- [23] Massimo Paolucci, Dirk Kalp, Ananddeep S. Pannu, Onn Shehory, and Katia Sycara. A planning component for retsina agents. In *Lecture Notes in Artificial Intelligence, Intelligent Agents VI*, 1999.
- [24] Ivan Porres. Rule-based update transformations and their application to model refactorings. *Software and System Modeling*, 4(4):368–385, 2005.
- [25] Shane Sendall and Wojtek Kozaczynski. Model transformation: The heart and soul of model-driven software development. *IEEE Software*, 20:42–45, 2003.
- [26] Rational Software. The rational unified process, 1998.
- [27] Jonathan Sprinkle, Aditya Agrawal, Tihamer Levendovszky, Feng Shi, and Gabor Karsai. Domain model translation using graph transformations. In *ECBS*, pages 159–167. IEEE Computer Society, 2003.
- [28] Ali Murat Tiryaki, Erdem Eser Ekinci, and Oguz Dikenelli. Refactoring in multi agent system development. *Lecture Notes in Artificial Intelligence*, 5244:183–194, 2008.
- [29] Ali Murat Tiryaki, Sibel Öztuna, Oguz Dikenelli, and Riza Cenk Erdur. Sunit: A unit testing framework for test driven development of multi-agent systems. In *AOSE*, volume 4405 of *Lecture Notes in Computer Science*, pages 156–173. Springer, 2006.
- [30] Cornelia Triebig and Franziska Klugl. Refactoring of agent-based simulation models. In *Multikonferenz Wirtschaftsinformatik*, 2008.
- [31] M. Williamson, K. Decker, and K. Sycara. Unified information and control flow in hierarchical task networks. In *Theories of Action, Planning, and Robot Control: Bridging the Gap: Proceedings of the 1996 AAAI Workshop*, pages 142–150, Menlo Park, California, 1996. AAAI Press.
- [32] Franco Zambonelli, Nicholas R. Jennings, and Michael Wooldridge. Developing multiagent systems: The gaia methodology. *ACM Trans. Softw. Eng. Methodol.*, 12(3):317–370, 2003.
- [33] Franco Zambonelli and Andrea Omicini. Challenges and research directions in agent-oriented software engineering. *Autonomous Agents and Multi-Agent Systems*, 9(3):253–283, 2004.

Programming social middlewares through social interaction types

Juan Manuel Serrano
 University Rey Juan Carlos
 C/Tulipan S/N
 Madrid, Spain
 juanmanuel.serrano@urjc.es

Sergio Saugar
 University Rey Juan Carlos
 C/Tulipan S/N
 Madrid, Spain
 sergio.saugar@urjc.es

Abstract—This paper describes a type-oriented approach to the programming of social middlewares. It defines a collection of metamodeling features which allow programmers to declare the social interaction and agent types which make up the program of a multiagent society for some application domain. These features are identified and formalised taking into account a specification of social middlewares as programmable, abstract machines. Thus, the proposed approach results in the type system of an interaction-oriented programming language. The paper uses the C+ action language and the CCALC tool as formal devices, so that metamodeling features are given formal semantics as new social law abbreviations which complement the causal law abbreviations of C+. This programming language approach contrasts with the common modeling approach endorsed by organizational methodologies, and promotes higher levels of formality and reusability in the specification of multiagent societies.

I. INTRODUCTION

Social middlewares are the responsible software infrastructures for the run-time management of software component interactions in computational societies. Unlike traditional object-oriented, service-based or messaging middleware approaches, social middlewares (e.g. AMELI [5], S-MOISE+ [11], MadKit [9], INGENIAS toolkit [8], etc.) provide software components with high-level *communicative* and *organizational* interaction mechanisms, which build to different extents on *normative* concepts such as empowerments, permissions, obligations, commitments, etc. It is claimed that the increased flexibility and expressiveness of these interaction mechanisms will result in a better management of component interactions in large-scale, multi-organizational, open distributed systems.

However, current social middlewares must overcome a number of shortcomings in order to achieve their full potential. Firstly, the set of generic interaction mechanisms which they are designed to support is not *extendable*, so that programmers are constrained to use the pre-defined abstractions (scenes, teams, groups, etc.) provided by the organizational metamodel of choice (e.g. ISLANDER [4], AGR [6], Moise+ [10], INGENIAS [13]). Secondly, applications can not be developed from generic, *reusable* modules which are specialised in the target application domain. Last, the run-time semantics of the organizational metamodels are not formally specified, which limits the *understandability* of the language constructs and the *portability* of the social middleware.

To address these limitations, this paper puts forward an approach to the programming of social middlewares which can be characterised along the following premises. Firstly, we build on a primitive and flexible notion of *social interaction* which attempts to provide the basic building blocks for the specification of any kind of communicative or organizational mechanism. Secondly, the social middleware is regarded as a *programmable machine* which is formally specified in technologically neutral terms as an abstract machine. Last, the behaviour of the social middleware is programmed through social interaction *types* which declare the characteristic structure and rules that govern the social interactions of the target application domain. The first two premises have been addressed in previous work, namely [15] and, respectively, [16]. The goal of this paper is to elaborate on the third premise. In particular, a set of metamodeling features for declaring social interaction types will be identified and their semantics formalised in terms of the underlying abstract machine. The chosen technique for formalising the overall approach is the action language C+ [7] and its accompanying tool CCALC [1]. In accordance with this election, metamodeling features will be represented as social law abbreviations which complement the standard set of causal laws provided by C+.

The rest of the paper is structured as follows. The next section briefly recalls the results shown in [16], namely it introduces social middlewares as programmable abstract machines. This section also describes, albeit briefly, the major features of the C+ formalisation and the conference management example used throughout the paper. Then, section 3 describes the partial set of metamodeling features of social interaction types, leaving their proper formalisation to the appendix of the paper. Finally, the paper concludes with a discussion of the major results with respect to competing paradigms, and a summary of current and future work.

II. SOCIAL MIDDLEWARE AS AN ABSTRACT MACHINE

The social middleware is in charge of managing the interactions between software components participating in a multiagent society. For instance, let's consider the development of an application to support the management of scientific conferences. In this setting, the social middleware is in charge of maintaining and driving the social processes which make up

the management of conferences (submissions, assignment and reviewing of papers, etc.). It does not deal with the actual job of software components that will engage in those processes at run-time¹, but with the management of processes themselves. This management is performed in accordance with the rules of conference management as specified by the types of social interactions that implement them. Before describing in the next section how to declare these types of interactions, this one summarizes the general structure of multiagent societies and the generic behaviour of the social middleware.

A. Social middleware structure

The interaction space of the middleware is shaped in terms of a tree of nested social interactions, so that the root of this tree represents the computational society being managed by the middleware. Besides social interactions, the computational society is made up of three major kinds of social entities: *agents*, *resources* and *social actions*. Agents and resources represent the two kinds of *roles* that can be played by software components attached to the social middleware within a particular social interaction [15]. These software components may be intelligent BDI components programmed with high-level languages (e.g. 2APL, Jason, etc.), or plain user interfaces (e.g. a web browser)². Resources represent non-autonomous software components which store information and/or provide computational services to the society. On the other hand, agents represent autonomous components which purport to achieve some goal. The activity of agents within the overall society can be decomposed into a role-playing tree of further agents deployed in arbitrary social interaction contexts.

In order to achieve its purpose, the activity of some agent eventually resolves itself into the performance of social actions, namely saying something to other agents (i.e. *communicative actions*—CAs), calling the services of computational resources (i.e. *invocations*) and seeing the state of social entities (i.e. *observations*). CAs, invocations and observations are *atomic* interaction mechanisms which take place within the context of social interactions. This paper exclusively focuses on CAs, particularly on those pre-defined CAs which make up the *standard CA library* of the language. This library includes, amongst others, the declarations *SetUp*, *Close*, *Join* and *Leave*. The *SetUp* CA allows agents to declare that a new interaction of certain type be initiated by the middleware; *Close* is used by agents to force the middleware to finish some ongoing interaction; last, *Join* and *Leave* allow agents to become members of some interaction and, respectively, abandon a role currently being played by them.

Figure 1 represents a run-time snapshot of a social middleware for conference management. The following graphical conventions are followed: social interactions instances are

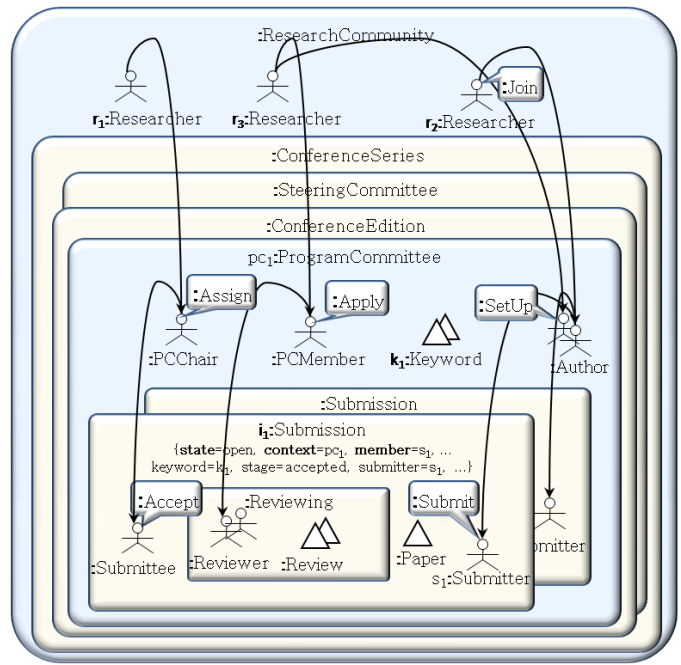


Fig. 1. Run-time snapshot of a multiagent society for conference management

represented by round corner rectangles, agents by stick figures and resources by triangles; last, speech bubbles represent the performance of communicative actions. A brief explanation of the different types of social entities involved follows in the next paragraphs:

- The root of the social interaction hierarchy is the *research community*, which is the context within which conferences actually take place. A given *conference series* is managed by its *steering committee*, which is responsible for the major issues concerning the different *conference editions*. The activity within a given edition is structured around the *program committee* (PC), *local committee*, etc. Within the PC, different *submissions* (one for each submitted paper) will take place, which in turn provide the context for the *reviewing* team in charge of evaluating that particular submission.
- The environments of the different interactions is made up of resources such as: the *paper* of submissions; the *reviews* uploaded within the reviewing team; the *calendar* and *keywords* created within the PC; and so forth.
- Three *researcher* agent roles and their corresponding role-playing hierarchies are shown in figure 1. The first one, r_1 , plays the role of *PC Chair* within PC pc_1 , which in turn behaves as *submittee* within the submissions of the PC. The second researcher, r_2 , participates in the PC as *author*, who in turn plays the *submitter* role in two different submissions. The third researcher, r_3 , participates in the PC both as *PC member* and *author*. As PC member it plays the role of *reviewer* in several submissions (typically, no more than three). Thus, agents may play several roles of different types within the same

¹In this application, software components attached to the social middleware will typically consist of plain user interfaces (e.g. a web browser).

²Thus, the use of *agent programming languages* for the implementation of *agent components*, i.e. components playing some *agent role*, is not mandatory. The reader is warned about the particular notion of *agents as roles* (vs. *agent as components*) endorsed by this paper [15, section 2].

kind of context, and roles of the same kind in different contexts of the same type.

- Figure 1 also illustrates the performance of several communicative actions (in particular, declarations). For instance, researchers become authors of PCs by *joining* those interactions to play a role of that kind. Once they are authors, they may *set up* a submission in order to *submit* a given paper. PC members may *apply* for certain papers in order to express their reviewing preferences. Papers will be definitely *assigned* for reviewing by the PC chairs. Eventually, the PC chair, as submittee of a submission, may *accept* the submitted paper, i.e. declare the paper as part of the conference program.

B. Social middleware dynamics

As far as the *dynamics* is concerned, two kinds of major forces which influence the evolution of the society can be considered: *external actions*, performed by software components over the middleware; and *internal triggers*, mainly related to the life-cycle of social entities. With respect to the latter, the social middleware is responsible for checking the conditions which signal that some interaction must be automatically *initiated* or *finished*; that some agent must be *played* or *abandoned*; etc. For instance, once a conference edition is initiated, a program committee is automatically initiated by the middleware. Similarly, a new reviewer role within a particular submission is automatically created for a given PC member when the PC chair assigns this agent the corresponding paper to review. As these examples illustrate, the life-cycle management of social entities mostly depends upon rules declared by its particular types, as the next section will show.

External actions are the means whereby software components may *enter* the society as an agent to participate in some interaction; *exit* the society as the player of some agent, thus abandoning any further role; or *attempt* one of its agents to perform a given social action. This paper exclusively focuses on this latter kind of external action. The processing of attempts by the social middleware is driven by *empowerments* and *permissions* rules. Empowerment rules establish which agents are institutionally capable of performing some social action. Permissions, on the other hand, establish the circumstances in which these powers may be exercised. For instance, any researcher is empowered to join a PC as author, but this action is only permitted within the submitting stage of the PC. If some agent is not empowered to do some action, the corresponding attempt causes no change at all in the institutional state; if some agent is empowered but not permitted to do the action, the forbidden attempt is registered accordingly; last, if the agent is both empowered and permitted, the action is executed by the middleware. For instance, the attempt of an author to *join* a PC in its submission stage causes the internal action *play* to execute, which in turn causes the creation of the corresponding agent role.

C. Formalisation in the action language C+

The specification of the social middleware has been formalised using the action language C+. The reader is referred to [16] for a detailed explanation of the following discussion. The action description which defines the abstract social middleware infrastructure is structured around a collection of generic, application-independent sorts, which encapsulate the common structure and dynamics of social interactions, agents, resources and social actions. Thus, the generic sort \mathcal{I} , whose specification is partially shown in figure 2, declares the fluents and action constants which characterise the state and dynamics of any kind of social interaction. These *standard* or pre-defined state parameters include the following fluents: *state*, which represents the execution state of the interaction (*none*-existent, *open* or *closed*); the boolean fluents *member*, *env* and *sub*, which represent the member agents, environmental resources and sub-interactions of a given interaction; and the statically determined fluents *context* and *initiator* which represent the interaction context of the interaction and the agent who set up the interaction (if it was not automatically initiated by the middleware). Figure 1 shows the values of some of these attributes for the submission i_1 . In particular, this interaction is open, its context is the PC pc_1 and has as member the submitter agent s_1 . Moreover, the figure also shows the values of other non-standard attributes which are characteristic of submission interactions: the *keywords* of the submission, its *stage* (accepted, in this case) and the *submitter* agent.

Figure 2 also shows the declaration of the action constants *initiate* and *finish*, together with the laws that define the preconditions and effects of the latter action. Thus, according to law 1, the action *finish* causes an interaction i to be closed; law 2 establishes that this kind of action can not be executed if the specified interaction i is not open; and law 3 declares that this action is not executed by default, leaving the specification of particular sufficient causes to application-dependent types (as will be described in the next section).

```

:- sorts
   $\mathcal{I}$ ;  $S_{\mathcal{I}}$ .
:- objects
  open, closed ::  $S_{\mathcal{I}}$ .
:- constants
  state( $\mathcal{I}$ ) :: inertialFluent( $S_{\mathcal{I}}+none$ );
  member( $\mathcal{I}$ ,  $\mathcal{A}$ ), env( $\mathcal{I}$ ,  $\mathcal{R}$ ), sub( $\mathcal{I}$ ,  $\mathcal{I}$ ) :: inertialFluent;
  context( $\mathcal{I}$ ) :: sdFluent( $\mathcal{I}+none$ );
  initiator( $\mathcal{I}$ ) :: sdFluent( $\mathcal{A}+none$ );
  ...
  initiate( $\mathcal{I}$ ,  $\mathcal{I}$ ), finish( $\mathcal{I}$ ) :: action.
:- variables
   $i$ ,  $i_c$ , ... ::  $\mathcal{I}$ .
/* laws */
...
finish( $i$ ) causes state( $i$ ) = closed.           (1)
nonexecutable finish( $i$ ) if state( $i$ ) ≠ open.    (2)
default ¬finish( $i$ ).                             (3)

```

Fig. 2. Partial specification of the generic social interaction type \mathcal{I}

III. PROGRAMMING THE SOCIAL MIDDLEWARE

Programming a multiagent society consists of specifying the social interaction types which model the relevant social processes of the target application domain. The specification of social interaction types involves in turn the specification of their member agent types and environmental resource types, as well as their characteristic types of CAs. Thus, the implementation of a multiagent society for conference management is made up of the types of social entities identified in figure 1: the social interaction types *ConferenceEdition*, *ProgramCommittee*, *Submission*, etc., and their accompanied environmental resource types (*Paper*, *Review*, etc.), member agent types (*Researcher*, *Author*, *Submitter*, etc.) and characteristic communicative action types (*Apply*, *Assign*, *Submit*, etc.).

In order to identify the metamodeling features which allow to declare the different types of social entities, it will be convenient to recall the three major ways in which the social middleware can be programmed. Firstly, the programmer may extend the set of standard attributes of social entities to account for the particular characteristics of the application domain. Secondly, the programmer may specify the particular conditions under which the middleware must create and destroy social entities (i.e. initiate and finish interactions, play and abandon agents, etc.). Last, the programmer may declare the empowerment and permission rules which drive the processing of social action attempts. Consequently, three classes of metamodeling features will be considered in the declaration of social entity types: *structural*, *life-cycle* and *attempt processing* features.

Formally, social entity types are defined using the subsort mechanism provided by the input language of CCALC [1, section 3] and the generic C+ sorts which implement the structure and behaviour of the social middleware [16]. Thus, the definition of a social interaction type proceeds, firstly, by declaring a new subsort of the generic interaction sort \mathcal{I} (figure 2); then, new fluent constants are declared which extend the definition of the generic sort; and, finally, new causal laws are provided which specify the structure and behaviour of the social middleware with respect to the new kind of social interaction. In particular, two kinds of causal laws can be used: those corresponding to the standard set of causal law abbreviations of the C+ language [7, appendix B]; and those defined by a new catalogue of *social law* abbreviations, which formalise the different metamodeling features of social entity types (partially listed and formalised in the appendix of this paper). Due to space limitations, the following subsections only introduce some of the devised social law abbreviations for the definition of social interaction and agent types.

A. Social interaction types

The metamodeling features of social interaction types will be illustrated with the specification of the *submission* interaction type, \mathcal{S} , shown in figure 3. To aid readability of the specification, social and causal laws are listed according to the life-cycle of interactions: firstly, those related to their

```

:- sorts
   $\mathcal{I} \gg \mathcal{S}; \mathcal{S}_{STAGE}$ .
:- objects
  submitted, accepted, rejected ::  $\mathcal{S}_{STAGE}$ .
:- constants
  /*inputs*/
  keyword( $\mathcal{S}, \mathcal{K}$ ) :: inertialFluent;
  /*outputs*/
  crc( $\mathcal{S}$ ) :: inertialFluent( $\mathcal{P}+none$ );
  /*local attributes*/
  stage( $\mathcal{S}, \mathcal{S}_{STAGE}$ ) :: inertialFluent;
  paper( $\mathcal{S}$ ) :: inertialFluent( $\mathcal{P}+none$ );
  /*aliases*/
  pc( $\mathcal{S}$ ) :: sdFluent( $\mathcal{PC}+none$ );
  submitter( $\mathcal{S}$ ) :: sdFluent( $SUBMITTER+none$ );
  submittee( $\mathcal{S}$ ) :: sdFluent( $SUBMITTEE+none$ );
  reviewingTeam( $\mathcal{S}$ ) :: sdFluent( $REVIEW+none$ ).
:- variables
  s ::  $\mathcal{S}$ .
/* laws */
/*Initiation laws*/
input keyword( $s, k$ ). (4)
context s is  $\mathcal{PC}$  alias pc. (5)
empowered a to setUp( $s, pc$ ) if member( $a, pc$ ). (6)
permitted a to setUp( $s, pc$ )
  if paperRegistration(calendar( $pc$ ))= $t$  &
      $t_{now}(clock) < t$ . (7)

/*Life-time laws*/
member s is  $SUBMITTER$  alias submitter. (8)
member s is  $SUBMITTEE$  alias submittee. (9)
environment s is  $\mathcal{P}$  alias paper. (10)
subinteraction s is  $\mathcal{REV}$  alias reviewingTeam. (11)
empowered submitter to submit. (12)
permitted submitter to submit
  if paperSubmission(calendar( $pc$ ))= $t$  &
      $t_{now}(clock) < t$ . (13)
...
/*Finishing laws*/
empowered chair( $pc(s)$ ) to close( $s$ ). (14)
permitted a to close( $s$ ). (15)
finish s if stage( $s, rejected$ ) ++
  state(submitter( $s$ ))= $abandoned$ . (16)
output crc( $s$ ). (17)
    
```

 Fig. 3. Submission interaction type \mathcal{S}

initiation; next, those pertaining to their life-time; and, finally, those concerning their finishing.

a) *Structural features*: The fluents declared for a new social entity type (i.e. not only social interactions) can be classified into four groups: *aliases*, *input*, *output* and *local* state parameters. Informally, the first group stands for those fluents which are introduced as aliases of standard fluents (e.g. *member*) to allow for more readable specifications. Input attributes are state parameters which must be set when the social entity is created. On the other hand, the meaning of output fluents directly refers to the destruction conditions of social entities: for instance, an interaction is automatically finished by the middleware when its output attributes are set. As for local fluents, these are normal C+ fluents whose meaning is established through common causal laws and

allow to simplify and improve the readability of other laws. The social law abbreviations `input` and `output` allow to specify newly declared fluent as input and output attributes, respectively; the `context`, `member`, `environment` and `subinteraction` abbreviations stand for the declaration of aliases pertaining to social interaction types. The formal definition of these abbreviations can be found in the appendix of this paper.

Thus, according to figure 3, a submission is modeled as a kind of process which receives as input a set of *keywords* (cf. law 4, which refers to variables s and k of the submission and keyword types, respectively), and has as goal the generation of the *CRC* (Camera Ready Copy) of a research paper (an output attribute, as declared by law 17). Thus, a submission can not be initiated without providing one keyword at least, and, as soon as the CRC of the paper is set the corresponding submission process will be automatically finished. On the other hand, the *stage* of the submission process is a local state parameter which holds the values *submitted*, *rejected* or *accepted*, as declared by the auxiliary \mathcal{S}_{STAGE} type. As for aliases, the member agents of a submission can be identified using the *submitter* and *submittee* fluents (besides the standard *member* fluent – see figure 2); similarly, the *paper* and *reviewingTeam* fluents stand for the environmental resources and subinteractions of the corresponding types; last, the *pc* fluent stands for the program committee context to which the submission belongs.

b) Life-cycle features: The specification of the generic social interaction type \mathcal{I} establishes that the execution of the *initiate* and *finish* actions are disabled by default (cf. law 3). Moreover, sufficient condition for the execution of these actions are absent from the specification. Therefore, programmers have to provide these conditions when defining a particular type of interaction \mathcal{T} , unless type \mathcal{T} is intended as an *abstract* type to be reused later in the definition of more specific types. Life-cycle conditions can be specified in two non-exclusive ways: directly, by defining new sufficient conditions for the *initiate* and *finish* actions; and indirectly, through the empowerment and permissions rules of the *SetUp* and *Close* standard CAs. The former approach allows the middleware to automatically initiate and finish interactions. The later provides agents with the possibility to force the execution of these internal actions. The social law abbreviations `initiate` and `finish`, defined in the appendix of this paper, allow to declare the social laws which allow the middleware to automatically govern the life-cycle of interactions.

For instance, submissions are only initiated if some author *sets up* an interaction of this kind within the program committee. Thus, these types of interactions are not automatically initiated by the middleware. On the contrary, a reviewing interaction is not set up by any agent but automatically initiated by the middleware when the stage of the submission is changed to *submitted*. Concerning finishing conditions, the submission process of some paper is automatically finished when its stage is set to *rejected* or the submitter agent is abandoned (law 16). A submission is also automatically finished when the CRC of

the paper is set by the submitter, as described above (law 17). Besides these “normal” ways of finishing a submission, the PC chair is also given extraordinary power to prematurely *close* a submission.

c) Attempt processing features: The definition of a new type of social entity may encompass the definition of new types of CAs which somehow aim at modifying the overall state of its instances. For example, the stage of submissions is set through the execution of particular CAs executed by the author and PC Chair agents, namely *Submit*, *Accept* and *Reject*. Besides these domain-dependent CAs, the *SetUp* and *Close* CAs also affect the execution state of social interactions. The attempts to perform any of these actions is subject to the their empowerments and permissions rules, which are absent from the generic specification. Therefore, the programmer is provided with two new social law abbreviations, `empowered` and `permitted`, which allow to govern the middleware behaviour with respect to the processing of CAs targeted at social entities of the new type.

For instance, social laws 6 and 7 declare the empowerment and permission rules for setting up a new submission process according to the requirements established above. Similarly social laws 14 and 15 declare the corresponding rules for prematurely closing a submission. Concerning the *Submit* CA, social laws 12 and 13 establishes (1) that the submitter is the only agent empowered to submit the paper of the submission; and (2) that permission to submit the paper is granted if the deadline for paper submission did not pass.

B. Agent types

Metamodeling features for defining agent types will be illustrated with the specification of the *submitter* agent type, partially represented in figure 4. Due to space limitations, the formal specification of the new abbreviations for social laws introduced in this section are skipped in the appendix. Similarly, the discussion of structural features is omitted.

d) Life-cycle features: The `play` and `abandon` social law abbreviations allow programmers to declare the particular rules which govern the automatic playing and abandonment of agents of the defined type. For example, the definition of the submitter agent type exploit the former abbreviation in social law 18, which establishes that a submitter agent is automatically created for a given author if that author is the initiator of the submission process and no submitter has already being created; the purpose of this new agent, as specified by law 19, is to set the CRC of the submission (i.e. to publish the submitted paper through the conference program). Concerning automatic abandonment conditions, the submitter agent type does not introduce any specific rule besides the ones declared by the generic agent type \mathcal{A} [16, section 5].

e) Attempt processing: The specification of empowerment and permission rules for new agent types employ the same abbreviations described in the last subsection for interaction types. The only difference lies in the kind of social actions pertaining to the specification: in this case the CAs *Join* and *Leave*. In the case of the submitter role, the creation of these

```

:- sorts
  A  $\gg$  SUBMITTER.
:- constants
  /*aliases*/
  submission(SUBMITTER) :: sdFluent(S+none);
  author(SUBMITTER) :: sdFluent(A+none);
  ...
:- variables
  submitter:: SUBMITTER.
/* laws */
/*Playing laws*/
play submitter for a within s
  if state(s)=open & initiator(s)=a &
     $\neg[\bigvee_{submitter} |submitter(s)=submitter].$  (18)
purpose submitter
  is  $[\bigvee_p |crc(submission(submitter))=p].$  (19)
  ...
/*Abandonment laws*/
empowered a to leave(submitter). (20)
permitted a to leave(submitter)
  if  $\neg$ stage(s, accepted). (21)

```

Fig. 4. Submitter agent type *SUBMITTER*

agents rely on the rules declared for automatic agent playing described above. Concerning its abandonment, author agents may prematurely leave one of its submitter roles, thereby causing the abandonment of the role and the cancellation of the submission (according to law 20). This power, however, may only be exercised if the paper has not already being accepted, as the permission law 21 specifies.

IV. DISCUSSION

This paper has put forward a *type-oriented* approach to the programming of social middlewares. Essentially, this approach is characterised by using types (of social interactions, agents, resources and CAs) as modules which encapsulate those structural and behavioural rules of the multiagent society which pertain to social entities of a certain kind. Moreover, the identification and formalization of the metamodeling features used in the declaration of social types strongly builds upon the specification of the social middleware as an abstract, programmable machine. The overall approach can be thus characterised as a *programming language* approach. We opted to call the resulting language SPEECH, given the relevance of CAs in the overall architecture of the language.

This *interaction-oriented* language contrasts with and complements common *component-oriented* languages such as 2APL, AgentSpeak, etc., aimed at the development of intelligent BDI agent components. Conversely, SPEECH is closely aligned with the attempt at designing a programming language for organizational artifacts reported in [17], [3]. In contrast with this work, however, we place agent components outside the realm of the social middleware, which helps to ensure their full autonomy and heterogeneity. Another significant difference is related to the nature of roles. In particular, the SPEECH specification of agent role types is devoid of any kind of computational feature, so that agent role instances just represent the *public interface* of agent components within the multiagent society. On the contrary, positions in [17] (i.e. agent

role instances) can execute plans to perform tasks delegated to them by their player agents. In fact, the specification of roles (i.e. agent role types) resorts to the typical constructs of BDI agent component languages. In our opinion, this blurs the distinction between agent components and agent roles, and undermines the separation of concerns between interaction and computation which lies at the heart of organizational programming languages. Last, besides agent roles the SPEECH language places a strong emphasis on social interaction types as a modularisation mechanism.

The SPEECH language is also closely related in spirit to common organizational metamodels for the specification of multiagent organizations such as ISLANDER [4], MOISE+ [12] and AGR [6]. Several methodological and conceptual differences, however, can be highlighted. Firstly, the programming language approach of SPEECH favours a higher degree of formality in the specification of the metamodel. Thus, in contrast to the common informal meanings of metamodeling constructs, the metamodeling features presented in this paper are grounded in the social middleware abstract machine presented in [16], and given formal semantics using the C+ action language. Thus, the proposal of this paper can be characterised as a first step towards the *type system* of a social interaction language, rather than as an organizational metamodel. Secondly, the SPEECH language places a strong emphasis on *specialisation* as a reusability mechanisms. Although some of the above-mentioned metamodels also support an inheritance relationship, this metamodeling features is at the core of the SPEECH specification. In fact, the operational semantics of the language is formalised through generic social types which are specialised in the specification of application-dependent types. In particular, the subsort mechanism of the C+ action language allows the programmer to override default laws of the super-sort, extend its signature with new fluents and actions, and/or refine the specification with new constraints. This strongly promotes the development of libraries of social types. For instance, an application-independent *submission* interaction may be defined, which could then be specialised for the particular case of paper submission. Similarly, other social interactions such as *invitations*, *discussion groups*, etc., may also be part of a generic library of social interactions, readily available for developers of arbitrary social process applications.

Current work deals with several extensions to the language to deal with obligations and sanctions (e.g. [3]), event processing, full-fledged communicative actions (e.g. [2]) and computational resources. Moreover, we aim to exploit the flexibility of the social interaction mechanism to define common metamodeling features of organizational metamodels (e.g. cardinalities and compatibility relations in MOISE+, performative structures in ISLANDER, etc.). In a more practical vein, current work also focuses on the implementation of a web-based social middleware for the language [14]. In this regard, the suitability of C+/CCalc for real deployments of multiagent societies is debatable. However, as a specification tool, they are of foremost importance to test in a techno-

logically neutral framework the features of the language. Moreover, they also provide invaluable help in the development of simple application prototypes. In particular, the conference management specification can be downloaded from <http://zenon.etsii.urjc.es/~jserrano/speech/apps/c+apps.tgz>.

REFERENCES

- [1] Varol Akman, Selim T. Erdogan, Joohyung Lee, Vladimir Lifschitz, and Hudson Turner. Representing the zoo world and the traffic world in the language of the causal calculator. *Artif. Intell.*, 153(1-2):105–140, 2004.
- [2] Guido Boella, Rossana Damiano, Joris Hulstijn, and Leendert van der Torre. A common ontology of agent communication languages: Modeling mental attitudes and social commitments using roles. *Applied Ontology*, 2(3-4):217–265, 2007.
- [3] Mehdi Dastani, Nick Tinnemeier, and John-Jules Meyer. A programming language for normative multi-agent systems. In Virginia Dignum, editor, *Multi-Agent Systems: Semantics and Dynamics of Organizational Models*, chapter 16. IGI Global, 2008.
- [4] Marc Esteva, David de la Cruz, and Carles Sierra. ISLANDER: an electronic institutions editor. In Maria Gini, Toru Ishida, Cristiano Castelfranchi, and W. Lewis Johnson, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'02)*, pages 1045–1052. ACM Press, July 2002.
- [5] Marc Esteva, Bruno Rosell, Juan A. Rodríguez-Aguilar, and Josep Ll. Arcos. AMELI: An agent-based middleware for electronic institutions. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, volume 1, pages 236–243, 2004.
- [6] Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizations: An organizational view of multi-agent systems. In *AOSE*, pages 214–230, 2003.
- [7] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories. *Artif. Intell.*, 153(1-2):49–104, 2004.
- [8] Jorge J. Gómez-Sanz, Rubén Fuentes-Fernández, Juan Pavón, and Iván García-Magariño. Ingenias development kit: a visual multi-agent system development environment. pages 1675–1676, 2008. May 12-16, 2008, Estoril Portugal.
- [9] Olivier Gutknecht and Jacques Ferber. The MADKIT agent platform architecture. *Lecture Notes in Computer Science*, 1887:48–55, 2001.
- [10] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Moise+: towards a structural, functional, and deontic model for mas organization. In *The First International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2002, July 15-19, 2002, Bologna, Italy, Proceedings*, pages 501–502. ACM, 2002.
- [11] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. S-moise+: A middleware for developing organised multi-agent systems. In Olivier Boissier, Virginia Dignum, Eric Matson, and Jaime Simo Sichman, editors, *Coordination, Organizations, Institutions, and Norms in Multi-Agent Systems*, volume 3913 of *LNCS*, pages 64–78. Springer, 2006.
- [12] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Developing organised multi-agent systems using the moise+ model: Programming issues at the system and agent levels. *IJAOSE*, 1(3/4):370–395, 2007.
- [13] Juan Pavón and Jorge Gómez-Sanz. Agent oriented software engineering with ingenias. In V. Marik, J. Muller, and M. Pechoucek, editors, *Proceedings of the 3rd International Central and Eastern European Conference on Multi-Agent Systems*. Springer Verlag, 2003.
- [14] Sergio Saugar and Juan Manuel Serrano. A web-based virtual machine for developing computational societies. In Matthias Klusch, Michal Pechoucek, and Axel Polleres, editors, *Cooperative Information Agents XII, 12th International Workshop, CIA 2008, Prague, Czech Republic, September 10-12. Proceedings*, volume 5180 of *Lecture Notes in Computer Science*, pages 162–176. Springer, 2008.
- [15] Juan Manuel Serrano and Sergio Saugar. Operational semantics of multi-agent interactions. In Edmund H. Durfee, Makoto Yokoo, Michael N. Huhns, and Onn Shehory, editors, *6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007), Honolulu, Hawaii, USA, May 14-18, pages 889–896. IFAAMAS*, 2007.

- [16] Juan Manuel Serrano and Sergio Saugar. Run-time semantics of a language for programming social processes. In Michael Fisher, Fariba Sadri, and Michael Thielscher, editors, *9th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA IX)*, volume 5405 of *Lecture Notes in Artificial Intelligence*, pages 37–56. Springer, 2009.
- [17] Nick Tinnemeier, Mehdi Dastani, and John-Jules Meyer. Roles and norms for programming agent organizations. In Decker, Sichman, Sierra, and Castelfranchi, editors, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pages 121–128, 2009.

APPENDIX

Abbreviation 1. An expression of the form “**input** $c(i, v) / c(i)$ ”, where c is a boolean (non-boolean) fluent constant, whose first argument i is a variable of an interaction sort I_d and whose second argument (result) is of sort V , stands for the following causal law, where i_c and v are variables of the interaction sort \mathcal{I} and sort V , respectively.

nonexecutable $\text{initiate}(i, i_c) \text{ if } \neg[\bigvee_v |c(i, v) / c(i) = v]$.

Thus, the resulting effect of declaring a new input parameter is the addition of a domain-dependent precondition to the *initiate* internal action (see figure 2).

Abbreviation 2. A set of expressions of the form “**output** $c_j(i)$ ”, where $j \in \{1 \dots n\}$, i is a variable of an interaction sort I_d , and c_j are non-boolean, optional constants of sort S_j , stands for the following action dynamic law, where v_j are variables of sorts S_j

caused $\text{finish}(i) \text{ if } \text{state}(i) = \text{open} \wedge \bigwedge_{j \in \{1 \dots n\}} [\bigvee_{v_j} |c_j(i) = v_j]$.

Thus, a set of expressions “**output** c_j ” implicitly establishes sufficient conditions for the execution of the *finish* standard action.

Abbreviation 3. A set of expressions of the form “**member i is** A_j **alias** f_j ”, where $j \in \{1 \dots n\}$, i is an interaction variable of sort I_d , A_j a collection of agent sorts, and f_j a set of boolean (non-boolean) binary (unary) fluent constants whose first argument is of sort I_d and its second argument (*optional*) result sort is A_j , stand for the following set of causal laws, where a_j are variables of sort A_j and a is an agent variable of sort \mathcal{A}

constraint $\text{member}(i, a) \rightarrow \bigvee_{j \in \{1 \dots n\}} [\bigvee_{a_j} |a = a_j]$.

caused $f_j(i, a_j) / f_j(i) = a_j \text{ if } \text{member}(i, a_j)$.

caused $\neg f_j(i, a_j) / f_j(i) = \text{none}$
if $\neg \text{member}(i, a_j) / \neg [\bigvee_{a_j} | \text{member}(i, a_j)]$.

Thus, the introduction of a new alias has also the intended meaning of constraining the types of agents that can be members of the interaction. The meaning of context, environment and subinteraction aliases can be similarly formalised.

Abbreviation 4. Let i be a variable of an interaction sort \mathcal{I}_d . The expression “**finish i if F**” stands for the action dynamic law:

caused $\text{finish}(i) \text{ if } F$.

Thus, the expression “**finish i if F**” is simply a wrapper of the corresponding action dynamic law which enacts the execution of the *finish* internal action. A similar abbreviation may be defined for declaring the automatic initiation of interactions.

Abbreviation 5. Let a and α be agent and institutional action variables. The expression “**empowered/permitted a to α if F**” stands for the static law

caused $\text{empowered/permitted}(a, \alpha) \text{ if } F$.

Thus, this social law abbreviation is just a wrapper of the static law which defines the predefined *empowered/permitted* fluent for the corresponding social action and agent sorts. The subexpression “if F” in the proposed abbreviations may be dropped if F is true.

Temporal Planning in Dynamic Environments for P-CLAIM Agents

Muhammad Adnan Hashmi
 Laboratoire d'Informatique de Paris 6
 Université Pierre et Marie Curie
 75016 Paris, France
 Email: Adnan.Hashmi@lip6.fr

Amal El Fallah Seghrouchni
 Laboratoire d'Informatique de Paris 6
 Université Pierre et Marie Curie
 75016 Paris, France
 Email: Amal.Elfallah@lip6.fr

Abstract—Time and uncertainty of the environment are very important aspects in the development of real world applications. Another important issue for the real world agents is, the balance between deliberation and reactivity. But most of the agent oriented programming languages ignore some or all of these important aspects. In this paper we try to fill this gap by presenting an extension to the architecture of CLAIM agent oriented programming language to endow the agents with the planning capability. We remove the assumption that agents' actions are instantaneous. We are interested in the temporal planning of *on the fly* goals. A coherent framework is proposed in which agents are able to generate, monitor and repair their temporal plans. Our proposed framework creates a balance between reactivity and deliberation. This work could be considered as a first step towards a complete temporal planning solution for an AOP language.

I. INTRODUCTION

Most of the agent oriented programming languages in the current literature use a PRS like approach to achieve the goals of agent. Some examples of these programming languages are Jason[1], 3APL[2], 2APL[3] and JACK[4]. But these languages lack the ability to incorporate planning. Sometimes the execution of the actions without planning results in the inability to achieve the goals. There has been some work to incorporate planning within such programming languages [5], [6], [7] but these systems do not take into account the duration of agent actions, neither do they consider the uncertainty of the environment. These systems assume that the agents' actions are instantaneous and that the effects produced on the environment are only those which are produced by the agent's actions. But these assumptions are unrealistic for the development of real world applications. There are some systems like ZENO[8], TGP[9], SAPA[10] which give the ability to plan with durative tasks and even there are systems which give this ability in the dynamic environments like IxTeT[11]. But these systems are separate planning solutions. They are not programming languages, so they lack the flexibility and control that a programming language offers to its programmers. Moreover, these systems are built on a proactive approach but in the real world applications it is necessary to create a balance between proactivity and reactivity because it is a dynamic world and the goals of agents are not necessarily given to him at the start, new goals arrive and some old goals are

dropped during the life cycle of the agent and some goals require immediate achievement. In this work, we try to fill these gaps by incorporating a temporal planner, an executor, an execution monitor and a plan repairing component to a CLAIM agent[12]. We call this extension of the language as P-CLAIM. The main problems dealt with in this work are 1) Modifications and extensions to the CLAIM agent's architecture to include a temporal planning component. 2) Execution monitoring and plan repairing. 3) Creating a balance between deliberation and reactivity.

In our proposed framework, we have made use of Hierarchical Task Network (HTN) planning technique. The main algorithm used to generate plan for a goal is JSHOP2[13], which is very efficient HTN planning system and plans for tasks in the same order that they will later be executed. The main motivation behind using the HTN planning technique is the similarities among the BDI model of agency and the HTN planning technique[14]. Due to these similarities HTN planning is more suitable and natural candidate for incorporating planning in a BDI like system.

The remainder of this paper is organized as follows. Section 2 puts some light on the current architecture of CLAIM language and JSHOP2 planner. In section 3, some important representations are presented which are helpful in understanding the agent architecture in P-CLAIM. Our proposed architecture of P-CLAIM agent with planning, execution and plan repairing components is presented in section 4. In section 5, we give an example to describe the working of system. Section 6 discussed some of the related work. Section 7 concludes the paper and some future directions are discussed.

II. BACKGROUND

In this section, we briefly discuss the architecture of CLAIM language and JSHOP2 algorithm to generate a plan. A multi-agent system in CLAIM is a set of distributed hierarchies of agents deployed on computers connected via a network. All the computers have a global clock. With respect to the hierarchical representation, an agent is a node in a hierarchy. It is an autonomous, intelligent and mobile entity. It has a parent and contains (optional) sub-agents, running processes and cognitive elements (e.g. knowledge, goals, capabilities). An agent can dynamically create another agent, and the newly

created agent becomes the sub-agent of the creator agent. In addition, an agent has three mobility primitives, *in* (enter another agent), *out* (leave another agent) and *move* (move from one hierarchy to another).

In CLAIM language, an agent can be defined as follows:

```
defineAgent agentName {
  parent=null | agentName ;
  knowledge=null | { (knowledge;)*}
  goals=null | { (goal;)*}
  messages=null | { (message;)*}
  capabilities=null | { (capability;)*}
  processes=null | { (process | )*}
  agents=null | { (agentName;)*}
}
```

For a more detailed description of CLAIM language, we refer to [12].

JSHOP2 is an HTN planning algorithm, and it deals with the procedural goals. Domain description required by JSHOP2 consists of methods and operators. A method indicates how to decompose a compound task into partially ordered subtasks. A method has three parts. The task for which the method is to be used, the condition which must be true in the current state to apply the method, and subtasks that need to be accomplished in order to accomplish that task. An operator is similar to the operators in classical planning and it tells how to perform a primitive task. It has a condition, a list of add effects and a list of delete effects. Planning proceeds by using the methods to decompose tasks recursively into smaller and smaller subtasks, until the planner reaches primitive tasks that can be performed directly using the planning operators.

The rationale behind choosing JSHOP2 for our work is threefold. Firstly, it is an HTN planner and the domain information from CLAIM can be easily transformed into the domain information needed by the planner due to the similarities among BDI like systems and HTN planning systems as discussed in [14]. Secondly, JSHOP2 plans for the actions in the same order that they will later be executed. So it knows the current state at every planning step. This property of the planner can be exploited for interleaving planning with execution and at every step planner can plan using the current state of the world. Thirdly, it can call external user defined functions to check the precondition of a method or an operator and this property is important for a planning component for CLAIM agents because in CLAIM language there could be calls to user defined functions to check the precondition of capabilities.

III. SOME IMPORTANT REPRESENTATIONS

In this section some important representations are presented which are helpful in understanding the architecture of an agent in P-CLAIM.

A. Domain Representation in P-CLAIM

We have modified the domain representation in CLAIM [12], in order to facilitate the translation to the representation needed by a planner. Agent's *capabilities* have now been

divided into *actions* and *activities*. *Actions* are the primitive actions that an agent can perform. Some of the *actions* are programmer defined while the others are already defined in the language like mobility primitives *in*, *out*, *move*. Programmer can also override the already defined *actions* to define his requirements more accurately. An action consists of a condition, a triggering message, the effects and a duration. *TriggerMessage(Act)* returns the triggering message of an action *Act*. Each effect of an action has an offset associated with it. This offset is the time taken by the action to produce the effect after the start of the action and it could be zero if this effect is achieved as soon as the action is started or it could be greater than zero. *Offset(Eff)* denotes the offset associated with an effect *Eff*. *Activities* are the short plans (recipes) in the plan library of the agent to achieve different composite goals.

B. Goal Representation in P-CLAIM

Goals in P-CLAIM are procedural goals. It means the goals of an agent are the tasks that agent wants to achieve. Some goals are initially given to the agent, when the multi-agent system is launched and some goals are given to the agent during the life of the agent using message passing by other agents or by user interaction. Goals have priorities associated with them. The priority of a goal could be Preemptive High, High or Normal. A goal having Preemptive High priority means that this goal should be immediately achieved by the agent, we also call this goal a reactive goal. High priority means that goal should be achieved before all the Normal priority goals currently present. Normal priority goals are the lowest priority goals. Goals with Preemptive High priority are stored in Global Reactive Goals (GRG) list and all other goals of agent are stored in a priority queue called Global Proactive Goals (GPG) list.

C. Messages Format

A message received by an agent in P-CLAIM has five parts. First part is the *identity*. Each message is assigned a unique number as identity. Second part is the *sender*, which represents the sender of the message. Thirdly, a message has a *priority* associated with it. This field has a value among Preemptive High, High and Normal. Fourthly, a message has a *proposition* which is the actual contents of the message. This *proposition* could be a new goal to achieve or it could be an information given to the agent which was demanded by the agent in an earlier message. Finally, a message has a *ResponseTo* field which is either blank or it contains a number pointing to the *identity* of an earlier message to which this message is responding.

D. Translation of Domain Description

The information needed by JSHOP2 algorithm to generate the plan includes the initial state information, goals information and domain description (methods and operators). In our formalism, this information is automatically extracted from the agent. Initial state information is taken from the *knowledge* of

the agent and from the hierarchical representation of MAS. Goal for the Planner is a one to one mapping from agent's goal to Planner's goal. In our framework, only one goal is passed to the JSHOP2 algorithm at a time. Agent's actions are mapped to the operators in JSHOP2. P-CLAIM agent's *activities* are converted into JSHOP2 methods. For each *activity* of the agent, an equivalent method is generated with the same name as that of *activity*. *Activity's* condition is mapped to the method's precondition. In JSHOP2, methods have subtasks. Subtasks may be primitive tasks or other composite tasks. Equivalently in P-CLAIM, the body of an *activity* consists of different processes. So we need to convert these processes into JSHOP2 methods and operators. To read in detail about this conversion, we refer to our earlier article[15].

E. Policy File

Each agent maintains a policy file in which it stores the importance of all other agents in the MAS for him. Importance of an agent depends on its position in the hierarchy relative to the position of the agent who is maintaining the policy. Importance also depends on the services provided by the agent during the life cycle of the agent. After receiving the message, the agent analyzes the policy file. Importance of the agent could be Normal or High.

IV. AGENT ARCHITECTURE

There are concurrently four threads running inside the agent all the time. In the following subsections, we explain these threads in detail. Figure 1 is showing the architecture of an agent.

A. Messages Handler

This thread is always waiting for the messages from other agents or from the user. It puts the messages into Planner Messages Queue(PMQ). These messages are either a request to achieve some goal or they are responses to some earlier sent message. After putting in the PMQ, these messages are fetched and analyzed. If the message contains some information demanded in an earlier message then this information is added to the *knowledge* of the agent along with an *acknowledgement* having the *identity* of the message in which this information was demanded. Agent's treatment of a message, which is a request to achieve some goal, depends on the priority associated with message and the importance of sender.

The Messages Handler fetches the goal attached with a message and assigns a priority to the goal based on the priority associated with message and the importance of sender. A goal fetched from a message of priority Preemptive High or High which is assigned by an agent having Normal importance in the policy file is assigned a High priority. It means that agent does not preempt his own goals for the goals assigned by an agent of Normal importance. A goal fetched from a message, sent by an agent having High importance in the policy file is assigned the same priority as of the message. After assigning a priority to the goal, the goal is added to one of the two global goals lists. A goal of priority Preemptive High is added to GRG list and a goal of priority High or Normal is added to GPG list.

There is another messages queue maintained inside the agent, called Executor Messages Queue(EMQ). Messages which are sent by the Planner for the execution of actions are put in the EMQ. These are the triggering messages for the actions in the plan generated by the Planner. Number of messages in EMQ are denoted by $length(EMQ)$ and $EMQ[i]$ denotes the i^{th} message in EMQ. Each triggering message in EMQ has a time stamp associated with it. $TimeStamp(Msg)$ denotes the time stamp associated with a triggering message Msg .

B. Planner

Planner starts when the multi-agent system is launched. Once started, the Planner procedure runs throughout the life cycle of the agent. When there are no goals in either of the goals lists then it sits idle and waits for new goals to arrive and as soon as a new goal arrives, it starts planning. Before starting the Planner, the agent goes through an initialization phase, in which it sets the values of certain global variables.

Three states of the world are maintained in the system, $SP(Act)$ which denotes the state of the world anticipated by planner just before the execution of the action Act , secondly SW is the current actual state of the world and $FinalSP$

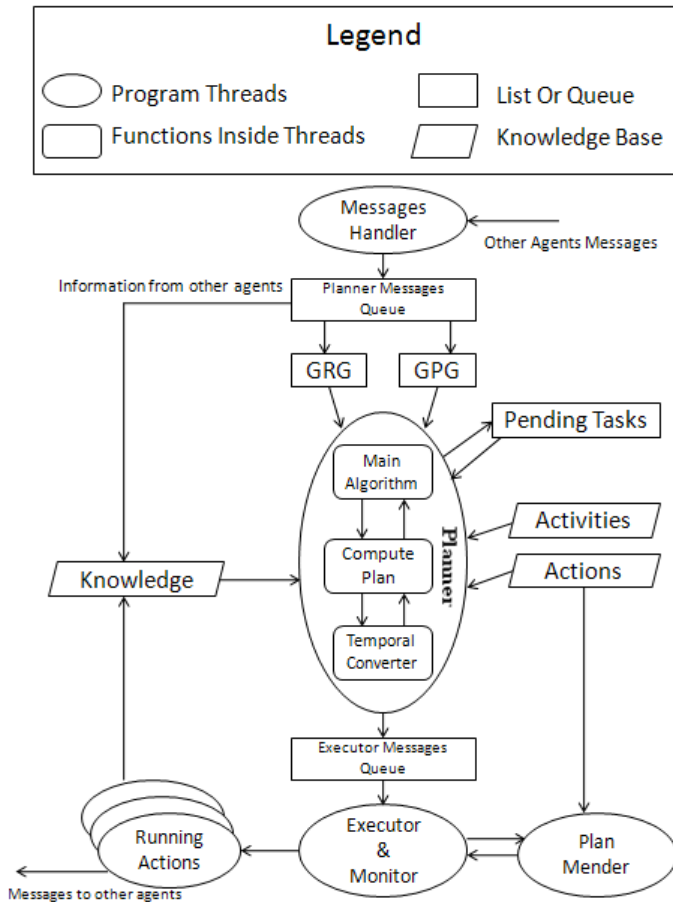


Fig. 1. A Running P-CLAIM Agent

Algorithm 1 *Main_Algorithm*

```

1: loop
2:   repeat
3:     Call Treat_Reactive_Goal
4:   until  $GRG = \phi$ 
5:   if  $GPG \neq \phi$  then
6:     Fetch first goal  $g \in GPG$ 
7:      $PPlan \leftarrow Compute\_Plan(FinalSP, g, D)$ 
8:     if  $PPlan \neq Fail$  then
9:       for  $i = 1$  To  $length(PPlan)$  do
10:         $TimeStamp(ExeMessages[i]) \leftarrow$ 
11:          $TimeStamp(PPlan[i])$ 
12:         $ExeMessages[i] \leftarrow TriggerMessage(PPlan[i])$ 
13:       end for
14:       Send ExeMessages to EMQ
15:     end if
16:   end loop

```

Algorithm 2 *Treat_Reactive_Goal*

```

1: Fetch first goal  $g \in GRG$ 
2:  $Suspension\_Signal \leftarrow ON$ 
3: Wait until  $\{Execution\_Signal = OFF\}$ 
4:  $Start\_Time \leftarrow$  Current system time
5:  $RPlan \leftarrow Compute\_Plan(SW, g, D)$ 
6: if  $RPlan \neq Fail$  then
7:   for  $i = 1$  To  $length(RPlan)$  do
8:     $TimeStamp(ExeMessages[i]) \leftarrow TimeStamp(RPlan[i])$ 
9:     $ExeMessages[i] \leftarrow TriggerMessage(RPlan[i])$ 
10:   end for
11:    $End\_Time \leftarrow$  Current system time
12:    $Duration \leftarrow End\_Time - Start\_Time$ 
13:   for  $i = 1$  To  $length(EMQ)$  do
14:     $TimeStamp(EMQ[i]) \leftarrow TimeStamp(EMQ[i]) +$ 
15:      $Duration$ 
16:   end for
17:   Send ExeMessages to EMQ
18: end if
19:  $Suspension\_Signal \leftarrow OFF$ 

```

denotes the state of the world to which the Planner has planned till now. More precisely, it is the state of the world anticipated by planner after the very last action that the Planner has planned for. In the initialization phase *FinalSP* is set equal to the *SW*. *Suspension_Signal* is set to *OFF* and *Execution_Signal* is set to *ON*.

The *Main_Algorithm* (Algorithm 1) runs in an infinite loop and ensures that reactive goals are immediately planned for and achieved. First it looks at the GRG list and if it is not empty, (Lines 2-4) the control moves to the procedure *Treat_Reactive_Goal* (Algorithm 2). Some of the notations used inside the *Treat_Reactive_Goal* procedure are as follows. $length(RPlan)$ denotes the number of actions in the plan *RPlan*. *ExeMessages* is an array of triggering messages for the actions in the plan. $TimeStamp(Act)$ denotes the time stamp assigned to an action *Act* for its execution. *Treat_Reactive_Goal* fetches the first reactive goal *g* and sets the *Suspension_Signal* to *ON* to ask the Executor to suspend the execution and waits for the *Execution_Signal* to go *OFF* which indicates that the Executor has suspended the execution (Lines 1-3) then it calls the *Compute_Plan* procedure to plan for the reactive goal (Line 5). The current

Algorithm 3 *Compute_Plan(S, G, D)*

```

1:  $P \leftarrow$  The Empty Plan
2:  $I \leftarrow S$ 
3:  $LG \leftarrow G$ 
4:  $LG_0 \leftarrow \{g \in LG : \text{no goal is constrained to precede } g\}$ 
5: loop
6:   if  $LG = \phi$  then
7:      $Plan \leftarrow Call\ Temporal\_Converter(I, P, D)$ 
8:     Return Plan
9:   end if
10:  Non deterministically choose any  $g \in LG_0$ 
11:  if  $g = Some\ Primitive\ Action$  then
12:    if  $g = Information\ Gathering\ Task$  then
13:      Generate and send message with identity  $x$ , for information retrieval to other agent
14:      Put all tasks depending on  $g$  in Pending Tasks list and assign them an identity  $x$ 
15:      Remove  $g$  from  $LG$ 
16:    else
17:       $A \leftarrow \{(a, \Theta) : a \text{ is a ground instance of an operator in } D, \Theta \text{ is a substitution that unifies } \{head(a), g\}, \text{ and } S \text{ satisfies } a's \text{ preconditions}\}$ 
18:      if  $A = \phi$  then
19:        Return Fail
20:      else
21:        Non deterministically choose a pair  $(a, \Theta) \in A$ 
22:         $S \leftarrow S + Add(a) - Del(a)$ 
23:        Append  $a$  to  $P$ 
24:        Modify  $LG$  by removing  $g$  and applying  $\Theta$ 
25:      end if
26:    end if
27:     $LG_0 \leftarrow \{g \in LG : \text{no other goal is constrained to precede } g\}$ 
28:  else
29:     $M \leftarrow \{(m, \Theta) : m \text{ is an instance of a method in } D, \Theta \text{ unifies } \{head(m), g\}, pre(m) \text{ is } True \text{ in } S, \text{ and } m \text{ and } \Theta \text{ are as general as possible}\}$ 
30:    if  $M = \phi$  then
31:      Return Fail
32:    end if
33:    Non deterministically choose pair  $(m, \Theta) \in M$ 
34:    Modify  $LG$  by removing  $g$ , adding  $sub(m)$ , constraining each goal in  $sub(m)$  to precede the goals that  $g$  preceded, and applying  $\Theta$ 
35:    if  $sub(m) \neq \phi$  then
36:       $LG_0 \leftarrow \{g \in sub(m) : \text{no goal in } LG \text{ precedes } g\}$ 
37:    else
38:       $LG_0 \leftarrow \{g \in LG : \text{no goal in } LG \text{ precedes } g\}$ 
39:    end if
40:  end if
41:  if New acknowledgement in knowledge then
42:     $id \leftarrow$  identity of the message whose acknowledgement has arrived
43:    Fetch all goals associated with message  $id$  from Pending Tasks list and put in the  $LG$ 
44:  end if
45:  if  $G$  is a proactive goal then
46:    repeat
47:      Call Treat_Reactive_Goal
48:    until  $GRG = \phi$ 
49:  end if
50: end loop

```

state of the world *SW*, the reactive goal just fetched *g* and domain description *D* are passed to *Compute_Plan* procedure. This procedure call returns a temporal plan *RPlan* for the reactive goal. Because every action in P-CLAIM is executed using a triggering message, so an array *ExeMessages* is generated containing the triggering messages for all the actions in the temporal plan *RPlan* with a *TimeStamp* associated with every message (Lines 7-10) and this array of

Algorithm 4 *Temporal_Converter*(I, P, D)

```

1: for  $j = 1$  TO  $no\_of\_literals(I)$  do
2:    $Production\_Time(Literal(I[j])) \leftarrow 0$ 
3: end for
4: for  $i = 1$  TO  $length(P)$  do
5:    $TimeStamp(P[i]) \leftarrow Max \{Production\_Time(Pre(P[i][j]))$ 
6:      $: j = 1$  To  $no\_of\_pre(P[i])\}$ 
7:    $SP(P[i]) \leftarrow$  World state anticipated before the execution of  $P[i]$ 
8:   for  $j = 1$  TO  $no\_of\_effects(P[i])$  do
9:      $Production\_Time(Literal(P[i][j])) \leftarrow TimeStamp(P[i])$ 
10:     $+ Offset(Literal(P[i][j]))$ 
11:   end for
12: end for

```

messages is sent to EMQ (Line 17) from where the Executor executes the actions triggered by these messages. But before sending *ExeMessages* to EMQ, the *TimeStamp* of all the messages currently in the EMQ is updated, because due to the suspension of execution, those triggering messages can not be executed at their intended time. So every message's *TimeStamp* is increased by the duration of the suspension (Lines 13-15). *Suspension_Signal* is then set to *OFF* (Line 18) to allow the Executor to resume execution and control is passed back to *Main_Algorithm* (Algorithm 1) which looks for another goal in GRG. The *Main_Algorithm* turns its attention to the proactive goals only when it finds that there is no reactive goal (Line 5). Algorithm fetches the first goal from GPG (Line 6). High priority goals are always fetched before Normal priority goals. Then *Compute_Plan* procedure is called with the parameters *FinalSP*, *g* and *D*. A plan *PPlan* is returned (Line 7) which is then sent to EMQ in the form of triggering messages (Lines 9-13). Now we elaborate the working of *Compute_Plan* procedure (Algorithm 3) (Many lines of the algorithm are taken from [13]).

Compute_Plan procedure is an extension of JSHOP2[13] algorithm. It takes three parameters *S*, *G* and *D* as input, where *S* is initial state, *G* is a list of goals and *D* is the agent's domain description. *Compute_Plan* procedure has an internal goals list called Local Goals (LG) list. Algorithm chooses a goal $g \in LG$ which has no predecessors (Line 4). At this point there could be two cases. The first case is if *g* is a primitive task, then procedure finds an operator *a* that matches *g* and whose preconditions are satisfied in *S*. It applies the action *a* to state *S* and adds it to his plan *P* (Lines 17,21-23). If no such operator *a* exists then procedure returns failure (Lines 18-19). In P-CLAIM a message to other agent is also treated as primitive action. So, *g* could be a message to other agent for information retrieval. If this is the case, then a message for the retrieval of information is generated with identity *x* and is sent to other agent. And all the tasks which depend on this information are put in the Pending Tasks list (Lines 11-15). All these tasks are assigned same identity *x* as of the message before sending them to Pending Tasks list.

The second case is where *g* is a compound goal, so a method needs to be applied for the decomposition of *g* into its sub-tasks. In this case the planner nondeterministically chooses a method instance *m* matching *g*, that decomposes *g* into sub-

goals (Line 29) and applies this method (Lines 33-34). If no such method *m* exists then procedure returns failure (Lines 30-32).

At the end of each planning step, the *Compute_Plan* procedure looks for any newly arrived *acknowledgement* for an earlier sent message. If a new *acknowledgement* for a message with identity *id* has been arrived then the procedure removes all the tasks depending on *id*, from Pending Tasks list and puts them in the Local Goals list to process those goals (Lines 41-44).

While planning for a proactive goal, the *Compute_Plan* procedure checks GRG for any new goals after each planning step and whenever it finds a goal in GRG, it suspends planning for the proactive goal and calls the procedure *Treat_Reactive_Goal*, which we have explained earlier (Lines 45-49). When GRG becomes empty, procedure resumes planning for the proactive goal from the same state at which it had suspended the planning. While planning for a reactive goal, the *Compute_Plan* procedure does not look at GRG, because a new reactive goal is treated only when all the previous reactive goals have been treated.

When *Compute_Plan* finds a plan for one goal, it converts the total order plan into a temporal plan by calling the procedure *Temporal_Converter* (Algorithm 4). The procedure takes three parameters *I*, *P* and *D*, where *I* is the initial state, *P* is the total order plan which is to be converted and *D* is the domain description file which is needed to extract the information about the durations of all the actions and offsets of all the effects. Some notations used in the procedure are as follows. $no_of_literals(I)$ denotes the number of literals in the initial state and $Literal(I[j])$ points to the j^{th} literal in initial state. $Production_Time(Lit)$ represents the time of achievement of a literal *Lit*. $length(P)$ returns the number of actions in the plan *P*. $no_of_pre(Act)$ and $no_of_effects(Act)$ denote the number of preconditions and number of effects of an action *Act* respectively while in the same vein $Pre(P[i][j])$ and $Literal(P[i][j])$ denote the j^{th} precondition and j^{th} effect of i^{th} action in plan *P* respectively. We have used a simple and efficient technique to convert a plan into temporal plan. The procedure starts by setting the *Production_Time* of all the literals in the initial state to 0 (Lines 1-3). Then procedure loops through all the actions starting from the first action, going towards the last one and sets the *TimeStamp* of the action to the maximum of the *Production_Time* of all its preconditions, because an action can be executed at least when all of its preconditions have been achieved (Lines 4-5). After setting the *TimeStamp* of an action, the procedure sets the *Production_Time* of all the effects of the action. The production time of an effect is the *TimeStamp* of the action plus the time at which the effect is produced by the action, the *Offset* of the effect (Lines 8-10).

C. Executor

The Executor is running in parallel with the Planner. It waits for triggering messages to come in the EMQ, fetches the messages and executes the actions associated with the

Algorithm 5 *Executor*

```

1: loop
2:   if Suspension_Signal = ON then
3:     Execution_Signal ← OFF
4:     Wait until {Suspension_Signal = OFF}
5:     Execution_Signal ← ON
6:   end if
7:   if EMQ ≠  $\phi$  then
8:     NextActions ← Fetch all next messages C from EMQ having
9:       the earliest TimeStamp from current system time
10:    NextTime ← TimeStamp(NextActions)
11:    Wait for system time to reach NextTime
12:    for i = 1 TO length(NextActions) do
13:      if All the actions in Prereq(NextActions[i]) has not sent
14:        acknowledgement for termination then
15:        Wait for all the acknowledgements
16:        Duration ← Time spent waiting for acknowledgements
17:        for i = 1 To length(EMQ) do
18:          TimeStamp(EMQ[i]) ← TimeStamp(EMQ[i]) +
19:            Duration
20:        end for
21:      end if
22:      if SP(NextActions[i]) = SW then
23:        Execute NextActions[i] in a separate thread
24:      else
25:        MPlan ← Plan_Mender(SW, SP(NextActions[i]))
26:        Execute MPlan
27:        for i = 1 To length(EMQ) do
28:          TimeStamp(EMQ[i]) ← TimeStamp(EMQ[i]) +
29:            TimeSpan(MPlan)
30:        end for
31:      end if
32:    end for
33:  end if
34: end loop
    
```

messages at their planned time stamps. Every running action sends an acknowledgement just before its termination to the Executor. Algorithm 5 is a simplified version of the Executor. The Executor fetches all the next messages from EMQ that have the closest *TimeStamp* to the current system time. Then the Executor waits for the system time to reach the *TimeStamp* of these messages (Lines 10-11). When system time approaches that time, the Executor checks whether the prerequisite actions of the actions associated with these messages have been terminated or not. If they have not been terminated then it waits for their termination. And increases the *TimeStamp* of all the messages in EMQ by the duration of waiting for their termination (Lines 14-20). Then it checks for any discrepancy among the current world state and the one anticipated by the Planner for the execution of these actions. If there is no discrepancy then these actions are executed in a separate thread (Lines 21-22) and the Executor fetches the next messages from EMQ. But if there is discrepancy among the two world states, then the Executor calls the Plan Mender to generate a plan from the current world state to the intended world state and executes the plan thus returned to remove the discrepancy (Lines 24-25). After executing this plan the Executor is ready to execute the actions which it had suspended due to discrepancy (Line 29). But before executing these actions, it augments their *TimeStamp* by the duration of the discrepancy removal.

Moreover, after executing each action, the Executor checks the *Suspension_Signal*. When *Suspension_Signal* is set to ON, it turns *Execution_Signl* to OFF, suspends the execution, and waits for *Suspension_Signal* to go OFF. The Executor resumes the execution once the *Suspension_Signal* is turned to OFF. But now the triggering messages for the plan of reactive goal are at the front of EMQ, so the Executor first executes the plan for the reactive goal for which it had suspended the execution and then it resumes the execution of plan on which it was working before the suspension (Line 29).

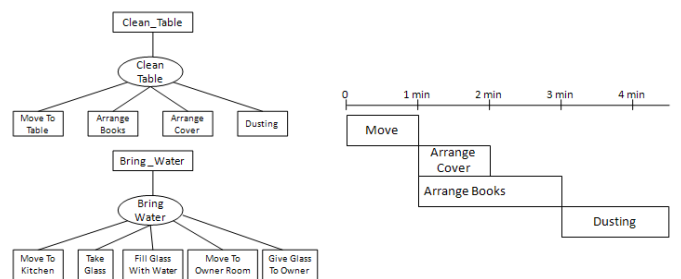
Algorithm 6 *Plan_Mender*(*I*, *G*)

```

1: Generate a plan P using SATPLAN from I to G ignoring the duration
2:   of actions
3: TP ← Call Temporal_Converter(I, P, D)
4: Return TP
    
```

D. Plan Mender

This procedure is responsible for repairing the plan. It takes as input the current actual world state *I* and the anticipated world state *G*. The Plan Mender generates a temporal plan for the agent to reach the anticipated world state starting from the current world state and returns this plan to the Executor. The Plan mender uses the classical STRIPS style planning technique to compute its plan because now the goals for the planner are a state to be reached (declarative goal). So, the Plan Mender just uses operators from the domain description file to compute the plan. In this case, the activities are not helpful in generating the plan which were used by the Planner component. The basic algorithm used by the Plan Mender is shown in Algorithm 6. Plan mender computes a plan without taking into account the durations of the actions using the SATPLAN planner[16] and then uses the procedure *Temporal_Converter* to convert the plan to a temporal plan.


 Fig. 2. (a).ROCO Activities (b).Plan for *Clean_Table*
V. EXAMPLE

In this example scenario we have one mobile agent ROCO, which is a home servant. When the MAS is launched then ROCO has the goal *Clean_Table*. ROCO has *activity* associated with this goal *Clean_Table*. All the *activities* of ROCO are shown in tree form in Figure 2(a) (A rectangle is showing a goal and an associated oval is an Activity associated with the goal, rectangles at the bottom without an

associated oval are Actions.). *Main_Algorithm* fetches goal *Clean_Table* from GPG list. And calls *Compute_Plan* to plan for this goal which generates a plan consisting of the following actions *Move(Room1,Table)*, *ArrangeBooks*, *ArrangeCover*, *Dusting*. A short description of the actions in this plan is shown in Figure 3.

<pre> Move(Room1, Table){ Duration = 1 minute Pre = At(ROCO, Room1) Effects = At(ROCO, Table) } </pre>	<pre> ArrangeCover{ Duration = 1 minute Pre = At(ROCO, Table) Effects = Arranged(Cover) } </pre>
<pre> ArrangeBooks{ Duration = 2 minutes Pre = At(ROCO, Table) Effects = Arranged(Books) } </pre>	<pre> Dusting{ Duration = 1.5 minutes Pre = At(ROCO, Table), Arranged(Books), Arranged(Cover) Effect = Clean(Table) } </pre>

Fig. 3. Description of actions in the temporal plan for *Clean_Table*

The plan is converted to the temporal plan using the procedure *Temporal_Converter* and the plan returned is shown in figure 2(b). In this example, all the effects of all the actions have an offset equal to the duration of the action. Here we explain the conversion of totally ordered plan to temporal plan. Procedure starts by assigning the *Production_Time* of all the literals in the initial state to 0. There is only one literal *At(ROCO, Room1)* in the initial state so *Production_Time(At(ROCO, Room1))* is set to 0. Now the procedure takes first action *Move(Room1, Table)* and sets its *TimeStamp* to 0, which is the maximum *Production_Time* from all of its preconditions. Next, the procedure sets the *Production_Time* of the effects of *Move(Room1, Table)*. This action has only one effect *At(ROCO, Table)*. *Production_Time(At(ROCO, Table))* is assigned the value *TimeStamp(Move(Room1))* plus *Offset(At(ROCO, Table))*. Putting the values, we get *Production_Time(At(ROCO, Table))* equals 1 minute, because *Offset(At(ROCO, Table))* is equal to the duration of *Move(Room1, Table)*. Now procedure moves to second goal which is *Arrange_Books* and sets its *TimeStamp* to be the maximum of *Production_Time* of all of its preconditions. It has only one precondition *At(ROCO, Table)* whose *Production_Time* has already been calculated to 1 minute. So *TimeStamp(Arrange_Books)* is assigned 1 minute. In this way the procedure continues and finds the plan shown in figure 2(b). Planner sends the messages for each action of the plan along with their *TimeStamp* to the EMQ for execution and Executor starts executing the plan. When the Executor has executed *Move(Room1, Table)*, *ArrangeBooks* and *ArrangeCover*, it checks that *Suspension_Signal* is set to *ON*, because the Planner has just fetched a reactive goal *Bring_Water* from the GRG. The Executor suspends the execution, sets the *Execution_Signal* to *OFF* and waits for the *Suspension_Signal* to go to *OFF* again. It receives the following plan in the EMQ, *Move(Table, Kitchen)*, *TakeGlass*, *FillGlassWithWater*, *Move(Kitchen, OwnerRoom)*, *Give(Glass, Owner)*. Now the Executor executes this plan.

After the execution of this plan *ROCO* is in *OwnerRoom*. Now the Executor resumes its suspended plan but before resuming the suspended plan, it increases the *TimeStamp* of all the actions in the suspended plan by the *TimeSpan* of the plan for goal *Bring_Water*, then it checks whether the preconditions of the suspended plan hold in the current state. The preconditions of its suspended plan are *At(ROCO, Table) ∧ Arranged(Books) ∧ Arranged(Cover)* and the current state is *Arranged(Books) ∧ Arranged(Cover) ∧ At(ROCO, OwnerRoom)*. The Executor calls Plan Mender to generate a plan from current state of the world to the intended state of the world. Plan Mender returns a plan consisting of only one action *Move(OwnerRoom, Table)*. Executor executes this plan, so *ROCO* moves to *Table*. Now again the Executor checks for any discrepancy among the current state and the anticipated state but now both states are same so the Executor executes the suspended plan i.e. it executes the *Dusting* action.

VI. RELATED WORK

In this section, we briefly review some work from the existing literature which is related to our work. Some of the research related to ours is CYPRESS[17], RETSINA[18], DECAF[19] and the systems proposed in [5] and [6].

In our opinion, the system closest to our research is CYPRESS system, which also integrates a planning system SIPE-2[20] with an execution system PRS[21]. It has the ability to react to the unanticipated changes in the environment by replanning and also deals with probabilistic planning. Our approach has the added advantage of handling temporal knowledge. Another aspect differentiating P-CLAIM to CYPRESS is the mobility of the agents. In P-CLAIM, the agents are mobile so the context of an agent changes while moving from one machine to another. The planner component must be able to deal with the changing context because the planning is interleaved with execution. An advantage of CYPRESS system over our proposed system is in the way CYPRESS performs replanning. We suspend the execution while computing a plan to remove any discrepancies. While CYPRESS system uses *asynchronous replanning* in which the system continues to execute the unaffected portion of the plan while a planning module computes a new plan.

Our system has many similarities with RETSINA. Like our system, RETSINA also interleaves planning with execution and supports planning for dynamic and real environments. But one main difference of RETSINA system with our system is that RESTINA system plans by only reduction of the top level task and it does not plan among the top level tasks, but our system uses a HTN planner which also plans among the top level tasks. So the plan generated is more optimal in our system than in RETSINA system. Another main difference is that RETSINA system does not use the existing information from the BDI system whereas our system proposes a method to use the existing agent's and world's information.

Another framework DECAF[19] which can be seen as an inspiration of RETSINA, relates to our system. But, in DECAF,

the planner only estimates preconditions, select task templates and instantiates them. It lacks the ability to anticipate future actions.

Like our system, [5] also provides a way to translate the information from a JACK[4] agent to the information needed by JSHOP[22] planner. Main differences of this approach with our approach are that in [5] it is the responsibility of the programmer to specify the points where the planner should be called while our system plans for each goal. Our system has the ability to deal with the unanticipated changes in the environment, while [5] has no such ability.

Another framework incorporating planning in a BDI language is presented in [6]. It incorporates classical planning into BDI framework. More precisely it extends the X-BDI[23] model to use the propositional planning algorithms for performing means-end reasoning. Our hypothesis is that our proposed system has the advantage of being more efficient as the HTN planning technique can find plans more rapidly with the help of additional domain knowledge provided by the programmer. Another important aspect is the loss of the domain knowledge provided by the programmer in [6]. The advantage of using the HTN planning is that the plans can be synthesized according to the intentions of the programmer without losing the domain knowledge.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have presented an extension to the CLAIM language to endow the agents with the capability to plan ahead. This modified and extended language is called P-CLAIM. Agents are able to create temporal plans. Execution monitoring and plan repairing components are added. A balance between deliberation and reactivity has been established and the agents are able to turn their attention while planning to the newly arrived reactive goals. This work can be considered as a first step towards a comprehensive temporal planning solution for an Agent Oriented Programming language.

After creating the temporal plan for an agent but before its execution, the plan of an agent should be coordinated with the plans of those agents with which the plan could be in conflict or whose plans could be helpful for this agent. Our next task is to propose a coordination mechanism to coordinate the temporal plans of different agents. Coordinating the plan of agent with every other agent in the MAS is very costly, so another important task to do is to intelligently calculate the set of those agents whose plan could be in conflict or whose plans could be helpful for the agent and then the plan should be coordinated with only those agents.

REFERENCES

- [1] R. Bordini, J. Hubner, and R. Vieira, "Jason and the Golden Fleece of agent-oriented programming," *Multiagent systems artificial societies and simulated organizations*, vol. 15, p. 3, 2005.
- [2] M. Dastani, M. van Riemsdijk, and J. Meyer, "Programming multi-agent systems in 3APL," *Multiagent systems artificial societies and simulated organizations*, vol. 15, p. 39, 2005.
- [3] M. Dastani and J. Meyer, "A practical agent programming language," *Lecture Notes in Computer Science*, vol. 4908, p. 107, 2008.
- [4] P. Busetta, R. Ronnquist, A. Hodgson, and A. Lucas, "Jack intelligent agents-components for intelligent agents in java," *AgentLink News Letter*, vol. 2, pp. 2–5, 1999.
- [5] L. de Silva and L. Padgham, "Planning on demand in BDI systems," *Proc. of ICAPS-05 (Poster)*, 2005.
- [6] F. Meneguzzi, A. Zorzo, and M. da Costa Mora, "Propositional planning in BDI agents," in *Proceedings of the 2004 ACM symposium on Applied computing*. ACM New York, NY, USA, 2004, pp. 58–63.
- [7] de Silva et al., "First Principles Planning in BDI Systems," in *Proceedings of the 8th international joint conference on Autonomous agents and multiagent systems*, S. Decker, Sichman and C. (eds), Eds., 2009, pp. 1105–1112.
- [8] J. Penberthy and D. Weld, "Temporal planning with continuous change," in *Proceedings of the national conference on Artificial Intelligence*. John Wiley & Sons Ltd., 1995, pp. 1010–1010.
- [9] D. Smith and D. Weld, "Temporal planning with mutual exclusion reasoning," in *International joint conference on artificial intelligence*, vol. 16. Lawrence Erlbaum Associates Ltd., 1999, pp. 326–337.
- [10] M. Do and S. Kambhampati, "Sapa: A domain-independent heuristic metric temporal planner," in *Proceedings of ECP-01*, 2001, pp. 109–120.
- [11] M. Ghallab and H. Laruelle, "Representation and control in IxTeT, a temporal planner," in *Proc. 2nd Int. Conf. on AI Planning Systems*, 1994, pp. 61–67.
- [12] A. El Fallah-Seghrouchni and A. Suna, "An unified framework for programming autonomous, intelligent and mobile agents," *Lecture notes in computer science*, Springer, pp. 353–362, 2003.
- [13] D. Nau, T. Au, O. Ilghami, U. Kuter, J. Murdock, D. Wu, and F. Yaman, "SHOP2: An HTN planning system," *Journal of Artificial Intelligence Research*, vol. 20, no. 1, pp. 379–404, 2003.
- [14] S. Sardina and L. Padgham, "Hierarchical planning in BDI agent programming languages: A formal approach," in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. ACM New York, NY, USA, 2006, pp. 1001–1008.
- [15] H. M. Adnan, "A Planning Component for CLAIM Agents," in *To appear in the Proceedings of International Workshop On Multi-Agent Systems Technology and Semantics*. IEEE Romania, 2009.
- [16] H. Kautz, B. Selman, and J. Hoffmann, "Satplan: Planning as satisfiability," in *5th International Planning Competition*. Citeseer, 2006.
- [17] D. Myers, L. Wesley, and A. Center, "CYPRESS: Reacting and Planning under Uncertainty," in *DARPA Proceedings: Rome Laboratory Planning Initiative*. Morgan Kaufmann, 1994, p. 111.
- [18] M. Paolucci, O. Shehory, K. Sycara, D. Kalp, and A. Pannu, "A planning component for RETSINA agents," *Lecture notes in computer science*, Springer, pp. 147–161, 2000.
- [19] J. Graham and K. Decker, "Towards a distributed, environment-centered agent framework," *Lecture notes in computer science*, Springer, pp. 290–304, 2000.
- [20] D. Wilkins, "Can AI planners solve practical problems?" *Computational Intelligence*, vol. 6, no. 4, pp. 232–246, 1990.
- [21] M. Georgeff and A. Lansky, "Procedural knowledge," *Proceedings of the IEEE, Special Issue on Knowledge Representation*, vol. 74, no. 10, pp. 1383–1398, 1986.
- [22] D. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila, "SHOP: Simple hierarchical ordered planner," in *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence table of contents*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1999, pp. 968–975.
- [23] M. Mora, J. Lopes, R. Vicari, and H. Coelho, "BDI models and systems: Reducing the gap," in *Proc. of ATAL-98, LNCS*, vol. 1555. Springer.

Agents Secure Interaction in Data driven Languages

Mahdi Zargayouna¹

¹ INRETS Institute,
Gretia Laboratory,
2, Rue de la Butte Verte
93166 Noisy Le Grand, France
Email: zargayouna@inrets.fr

Flavien Balbo^{1,2}

² University of Paris-Dauphine,
Lamsade-CNRS Laboratory,
Place du Maréchal de Lattre de Tassigny
75775 Paris, France
Email: balbo@lamsade.dauphine.fr

Serge Haddad³

³ École Normale Supérieure de Cachan,
LSV-CNRS Laboratory,
61, Avenue du Président Wilson
94235 Cachan, France
Email: haddad@lsv.ens-cachan.fr

Abstract—This paper discusses the security issues in data driven coordination languages. These languages rely on a data space shared by the agents and used to coordinate their activities. We extend these languages with a main distinguishing feature, which is the possibility to define fine-grained security conditions, associated with every datum in the shared space. Two main ideas makes it possible: the consideration of an abstraction of agents' states in the form of data at language level and the introduction of a richer interaction mechanism than state-of-the-art templates. This novel security mechanism allows both agents and system designers to prohibit undesirable interactions.

I. INTRODUCTION

When designing logically distributed applications and open Multi-Agent Systems (MAS), developing applications without knowing either the overall structure of the system or the agents that will be functioning in it is a challenge. Data driven coordination languages, with the pioneer language Linda [1] and its extensions, provide a great deal of flexibility and are a promising approach to meet this challenge. These languages are based on the notion of a shared data repository composed of data used by the agents to interact and to synchronize their activities. Agents communicate by exchanging tuples via an abstraction of an associative shared memory called the *tuplespace*. A tuplespace is a multiset of tuples (tuples duplication is allowed) and is accessed associatively (by content) rather than by address, by specifying a template. Every tuple is a sequence of one or more typed values and every template is a sequence of one or more typed values or formal fields. Every tuple field matches with the corresponding template field if they have the same value or are of the same type.

Relying on a shared space for agent interaction naturally handles open systems design [2]. The advantage is that they provide the possibility for new agents to join the system and, since all the agents have a common interlocutor (the shared space), they don't have to manage an up-to-date address book of the other agents of the system. Nevertheless, the openness management implies a secure relationship between the agents and the shared memory. The data driven coordination model has to deal with the following security threats [3]: 1) threat on authenticity; 2) threat on confidentiality; 3) threat on availability. A threat on authenticity occurs when an agent acts instead of another agent. The data driven coordination model is designed to promote anonymous interaction, but if the tuples contain values related to the agents that insert, read or consume

it then the authenticity of these data has to be validated. For example, in [4] the authors present a messaging service where the interaction between the agents is mediated by a broker that is grounded on a tuplespace. In this application, the authenticity of the agents related to a message exchange has to be guaranteed. The confidentiality threats are related to the interception by an agent of another agent's confidential information or message. Following the data driven approach, any agent can read/remove any tuple stored in the tuplespace simply by exploiting formal fields (variables) [1], which act as wildcards [5]. Therefore, a template having two wildcard fields can be used to read or remove any tuple containing two data fields [4]. The threats on availability concern the consequence of the deletion and insertion operations on the behavior of the tuplespace. The deletion of information is a consequence of the lack of confidentiality and implies that it is not possible to guarantee the correct behavior of the system. The threat related to the addition of information concerns malicious agents that can insert an unbounded number of tuples; in such a way, since the manager of the space has to handle any tuple's insertion operation, a process can generate a denial of service attack [4].

In data driven coordination languages, security is generally enforced by using multiple (logical) spaces or by stating "Interaction Laws". With multiple spaces (e.g. Klaim [6], RBAC and TucSon [7], SecSpaces [8] and SecOS [9]), two agents that wish to exchange confidential data use a space that is known only by the two of them. However, security with multiple spaces is defined in a coarse-grained way, since accessing one space gives the possibility to access all of its data, and being excluded from one space means not having access to any of its data. The laws in LGI [10] allow data to be secured by specifying conditions on the states of the agents and the content of the data. Tuple-space reactions are associated with agents' actions so as to always result in a coherent configuration. However, agents cannot manage the security of the data they add, and only the designer can specify security conditions.

We would like to equip data driven coordination languages with a security mechanism that allows for the protection of the exchanged data in a fine-grained way. We want to let agents specify, when they add a datum to the data space, the conditions under which it can be read or taken by others.

We also would like the designer of the system to specify the conditions under which an agent can or cannot add a certain datum to the space, following the application logic. To do so, we perform several modifications of the shared space model, and propose a new language, called LACIOS (Language for Agent Contextual Interaction in Open Systems), which is the linguistic embodiment of the modified model.

In [5], the authors classify secure data driven languages into entity driven and knowledge driven languages. The idea behind the knowledge oriented approach is that tuple spaces, tuples or single data fields are decorated with additional information and an agent can access the resources only in the case they prove their knowledge of these information. In the case of the entity oriented approach, additional information associated to resources list the entities which are allowed to access the resources. Our proposal can be classified as an entity oriented approach. However, instead of listing the agents that can access a datum, in LACIOS these agents are described *symbolically*, i.e. their properties are defined without pointing them namely.

The remainder of this paper is structured as follows. We give an overview of LACIOS in section II. Section III gives the basic syntax of our language; Section IV addresses the security issue, and section V provides the complete specification of agents' behaviors. Section VI presents the programming language JAVA-LACIOS. The proposal is discussed with respect to the state of the art in section VII before concluding with further lines of research.

II. OVERVIEW OF LACIOS

A MAS written in LACIOS is defined by a dynamic set of *agents* interacting with an *environment*, which is composed of a dynamic set of *objects*. To illustrate the syntax of LACIOS, an example application is used throughout the paper. In this example, human travelers are in a train station in which schedules, booking, payment services and information sources coexist. Two agent types are considered in here: Traveler agents represent travelers wishing to make a journey and Train agents represent trains, and generate information concerning future departures, arrivals, delays, etc. All these agents interact by exchanging data via a shared space in the same way as for all data driven coordination models.

A MAS written in LACIOS is an open system in two ways. As for every data driven language, agents in LACIOS can join and leave the system freely. In addition, external - non modeled - systems and users can interact with the MAS. As we will define it later, users (e.g. travelers) interact with the MAS by instantiating the values of certain variables in the code of the agents that represent them. External systems (e.g. trains) can interact with the MAS by instantiating variables with values as well. They can also execute agents that interact with the MAS Environment directly. The figure 1 illustrates the MAS architecture. The modeled MAS executes on a host, where (local) agents add, read and take objects to/from the MAS environment. Every agent is either independent (like agent 1), or representing a non-modeled system/user in the MAS.

The agents that are defined in a LACIOS program are usually the *local* agents. The users, external agents and external systems that are represented by an agent in the MAS are not modeled, only their actions are observed in the MAS, through the nondeterministic behavior of the local agent. An agent in LACIOS is then an entity, that has a state, a local memory and a nondeterministic behavior. As we will define it later, the whole behavior of the agent is not defined in LACIOS. An agent can have a complex behavior, by using additional operators, besides the standard operations defined in LACIOS.

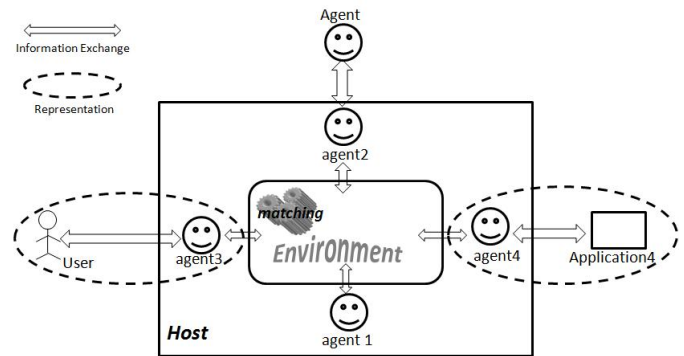


Fig. 1. LACIOS Architecture

Since agents in LACIOS don't interact directly, but via the environment, our definition of an agent is close to the general definition given by [11]:

Definition 1 (Agent): An agent is a computer system capable of autonomous action in some environment in order to meet its design objectives.

From the security point of view, LACIOS has two objectives: 1) to support a global control by the environment of the insertion by the agents of objects in order to ensure that the new objects are not fraudulent (authenticity, availability), 2) to support a local control by the agents that can specify *who* can access the object that they add to the environment in order to ensure their privacy (authenticity, confidentiality and availability). To do so, agents have to have a state defining who they are. This is the first modification we perform to the original model: the consideration of an abstraction of agents' states in the form of data at language level. These states are defined as a set of *property*←*value* pairs (e.g. $\{identifier \leftarrow 10, position \leftarrow node_1\}$). Agents' states in LACIOS are data representing the state of the agents that are accessed by the environment only for matching and security purposes (they are not directly accessible by the other agents).

III. BASIC SYNTAX AND INFORMAL SEMANTICS FOR LACIOS

LACIOS is a data driven language for the design and implementation of open and secure MAS. For the specification of agent behavior, four primitives inspired by Linda and a set of operators borrowed from Milner's CCS [12] have been used. An MAS written in LACIOS is defined by a dynamic set of *agents* interacting with an *environment* - denoted Ω_{ENV} ,

which is composed of a dynamic set of *objects*. Figure 2 illustrates the general principle of LACIOS. Agents are defined by a behavior (a process), a state (data) and a local memory in which they store the objects they perceive or retrieve from the environment. Agents can *perceive* (read only) and/or *retrieve* (read and take) objects from the environment. First the four primitives of LACIOS will be presented; their parameters will be defined along with the details of the language.

$$\mu ::= \text{spawn} \mid \text{add} \mid \text{update} \mid \text{look}$$

The primitive *spawn* launches a new agent and provides it with an initial state and a behavior. An *add* action adds an object to the environment. The *update* primitive changes locally the old values of the agent's state to the new ones.

Unlike traditional retrieval primitives, the *look* primitive enables agents for both the perception and retrieval of objects as will be described below. The primitive looks for objects in the environment that satisfy the agents' conditions expressed as parameters. Agents can use their own states in the parameter expression of a *look*, which are accessible by the environment only, when the parameter expression is evaluated. Note that, the state of an agent cannot be accessed directly by the other agents (through a *look* expression). In order to be observable to the others, an agent has to add an object representing itself to the environment autonomously (as in Fig. 2, where the agent decides not to publish a part of its state). Having data representing agents in the environment allows the agents of the system to discover each other by simply interrogating the environment *à la Linda*.

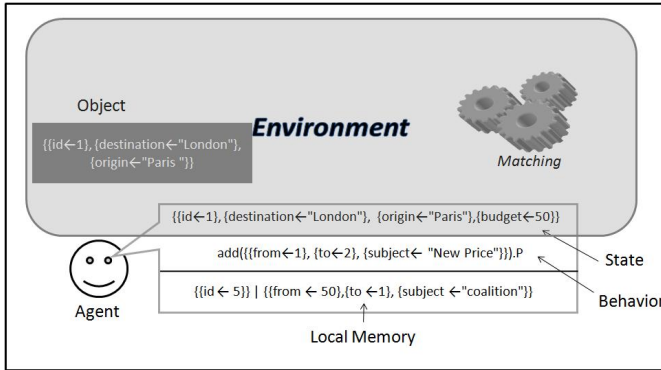


Fig. 2. General Schema

A. Data structure

For LACIOS, we define a standard information system data structure: every item of data in the system has a *description*, i.e. a set of *property*–*value* pairs, and all the properties of the language are typed. The notions of type, property and description are defined as follows.

Definition 2 (Types): The types of the language are defined as $type_1, \dots, type_{nbt}$. Every $type_i$ is a set such that $\forall(i, j) \in \{1, \dots, nbt\}^2, i \neq j, type_i \cap type_j = \{nil\}$

Remark 1: We assume the existence of the *boolean* type in the language, i.e. $\exists i \in \{1, \dots, nbt\}, type_i = \{\text{true}, \text{false}, \text{nil}\}$

Notation 1: We denote the set of values supported by the language as $\mathcal{T} = \bigcup_{i=1}^{nbt} type_i$.

Definition 3 (Property): \mathcal{N} is the property space, and is a countable set of properties. A property $\pi \in \mathcal{N}$ is defined by a type $type(\pi) \in \{type_1, \dots, type_{nbt}\}$.

The value *nil* has a twofold use in the syntax of LACIOS. First, it represents every semantic error in a program. When a semantic error is encountered, the corresponding expression is set at *nil*. Second, a property whose value is equal to *nil* is considered as undefined (as if it is nonexistent), and is usually omitted.

Notation 2: We note $unknown_\pi$ a value of the type $type(\pi)$ that is defined but doesn't have a value. For instance, $unknown_{destination}$ is a value of the same type as the property *destination*, whose value is (temporarily) unknown.

A description is composed of properties and their corresponding values.

Definition 4 (Descriptions): \mathcal{DS} is the set of descriptions. A description is a function that maps properties to values, i.e. $d \equiv \{\pi \leftarrow v_\pi \mid v_\pi \in type(\pi)\}_{\pi \in \mathcal{N}}$. The mapping is omitted when $v_\pi = nil$. We use $d(\pi)$ in order to access the value v_π . For each description, the set of properties $\{\pi \mid d(\pi) \neq nil\}$ is finite.

In LACIOS, each description is associated with an *entity*, which can be an *object* or an *agent*. Objects are defined by their descriptions (\mathcal{O} is the set of objects), while each agent is defined by a description (their state), a behavior and a local memory (\mathcal{A} is the set of agents).

For instance, let o_1 be an object representing a traveler, d_{o_1} could be defined as follows: $\{id \leftarrow "o1", destination \leftarrow "London", origin \leftarrow "Paris"\}$. In this example, $d_{o_1}(origin)$ is equal to "Paris".

Definition 5 (Entities): $\Omega = \mathcal{A} \cup \mathcal{O}$ is the set of entities of the MAS. Each entity $\omega \in \Omega$ has a description as defined above denoted by d_ω . The value of the property π of the entity ω is denoted by $d_\omega(\pi)$.

Remark 2: We assume the existence of the type *reference* in LACIOS, a value of the type *reference* designates an entity in Ω , i.e. $\exists i \in \{1, \dots, nbt\}, type_i = \Omega \cup \{nil\}$.

B. Expressions

Expressions are built with values, properties and operators, and are used by agents to describe the data they handle, either locally or to interact with the environment.

Definition 6 (Operators): Each operator *op* of the language is defined by:

- (i) $arity(op)$ The number of parameters of the operator,
- (ii) $par(op) : \{1, \dots, arity(op)\} \rightarrow \{1, \dots, nbt\}$, $par(op)(i)$ gives the index of the type of the i^{th} parameter of the operator *op*,
- (iii) $ret(op) \in \{1, \dots, nbt\}$, the index of the type of the value resulting from the evaluation of *op*.

For instance, let $type_1 \equiv \text{boolean}$. The operator and is defined as follows:

$arity(\text{and}) = 2$, $par(\text{and})(1) = par(\text{and})(2) = 1$ and $ret(\text{and}) = 1$.

Besides basic operators, additional operators can be defined by the programmer, specifying complex agents' processes. LACIOS is then used mainly for coordination purposes, while the computational model remains non modeled.

An expression may simply be a value, an operator, or a property. For instance, $destination \neq \text{"Paris"}$ is a (boolean) expression. If an expression is a property, it refers to a property of the agent that is evaluating it. For instance, when $destination$ appears in the behavior of agent a as in the example above, it designates the destination of a . If a property $companion$ of agent a is of the type *reference*, $companion.destination$ designates the destination of the *companion* of a .

Definition 7 (Expressions): Exp is the set of expressions. An expression $e \in Exp$ is generated via the grammar found in Table I.

$e ::= nil$	
v	, with $v \in \mathcal{T} \setminus nil$
π	, with $\pi \in \mathcal{N}$
$op(e, \dots, e)$, with op an operator of the language, and nil doesn't appear in any e
$\pi.e$, with $\pi \in \mathcal{N}$ and $type(\pi) = \Omega$

TABLE I
SYNTAX OF AN EXPRESSION

In a description, an agent can associate an expression with a property, instead of a value. The result is a *symbolic description* which is transformed into a description when its associated expressions are evaluated.

Definition 8 (Symbolic descriptions): SDS is the set of symbolic descriptions. A symbolic description $sds \in SDS$ is a description that maps properties π to expressions e_π , i.e. $sds \equiv \{\pi \leftarrow e_\pi \mid type(e_\pi) = type(\pi)\}_{\pi \in \mathcal{N}}$.

Below is the definition of the *add* and *update* primitives, together with their symbolic descriptions parameters.

$$\mu ::= \dots \mid add(sds) \mid update(sds)$$

In order to add an object, an agent executes $add(sds)$, and an object whose description is the evaluation of sds is added to Ω_{ENV} . For instance, $add(\{ticket_{id} \leftarrow ticket.id, price \leftarrow ticket.price + 20, owner \leftarrow id\})$ adds an object to the environment whose *owner* is equal to the *id* of the calling agent, $ticket_{id}$ is equal to the property *id* of an object in the memory of the calling agent, referred to by the property *ticket*, and whose *price* is 20 more than the price paid by the agent. The primitive $update(sds)$ updates a set of properties of the agent with the evaluation of the expressions in sds . When $update(sds)$ is executed, the value of every property π in sds becomes equal to the evaluation of the corresponding expression e_π . For instance, if an agent a executes $update(\{budget \leftarrow budget - 20, destination \leftarrow$

*"Budapest" \}), its *budget* is decreased by 20 and its *destination* becomes "Budapest".*

C. Matching

Since we consider a data structure richer than tuples, we also use a matching mechanism richer than templates. To do so, the expressions' syntax is enhanced with *variables*, which designate objects not known by the agent, but which will be discovered during the matching process and will be replaced by objects from the environment before their evaluation. Below is the definition of a variable.

Definition 9 (Variables): \mathcal{X} is the set of variables. A variable $x \in \mathcal{X}$ is defined by its type $type(x) \in \{type_1, \dots, type_{nbt}\}$.

The syntax of an expression becomes:

$$e ::= \dots \mid x.e \text{ with } x \in \mathcal{X} \wedge type(x) = \Omega$$

For instance, consider the following boolean expression e : $t.destination = \text{"London"} \wedge t.price \leq budget$. In this boolean expression, t designates an object, unknown for the moment, where the property *destination* of t has to be "London" and the *price* has to be less than the *budget* of the agent for the expression to be evaluated to true. In this case, the agent executing *look* with e as a parameter will perceive or retrieve the object.

We can now provide the complete definition of the primitive *look*.

$$\mu ::= \dots \mid look(sds_p, sds_r, e)$$

We choose to use a single primitive to access the environment. The primitive $look(sds_p, sds_r, e)$, with sds_p and sds_r symbolic descriptions, allows both object perception and retrieval (perception and removal from Ω_{ENV}). It blocks until a set of objects C becomes present in Ω_{ENV} such that the expression e is evaluated to true. When an agent executes $look(sds_p, sds_r, e)$, the set of objects of the environment C is selected for matching with e (each variable is unified with an object from C). The expression e has to be evaluated to true with this unification for *look* to be executed. The objects associated with the variables in sds_p are perceived and those associated with the variables in sds_r are retrieved. For instance, the following instruction: $look(\{train \leftarrow tr\}, \{ticket \leftarrow tk\}, tr.destination = \text{"London"} \wedge tk.price \leq budget \wedge tk.train = tr.id)$ looks for two objects that will be unified with tk and tr . The object associated with tr will be perceived while the object associated with tk will be retrieved. After this instruction has been executed, the two objects will be present in the local memory of the caller agent, which will have two additional properties of the type *reference*: *ticket*, which refers to the object associated with the variable tk and *train*, which refers to the object associated with tr . The object unified with tk won't be present in Ω_{ENV} anymore.

D. Interaction with External Systems/Users

Consider an agent having two properties *destination* and *budget* that are unknown before the execution. The values of these properties come from an external system (e.g. a Web server, a GUI, etc). Here is the description of this agent which properties will be defined during execution resulting from their instantiation by an external system: $\{budget \leftarrow b, destination \leftarrow d\}$, where b and d are variables. Only the action of the external system will be observed, i.e. the assignment of values to the variables, while the system itself is not modeled. We enhance the syntax of an expression with free variables as follows:

$$e ::= \dots \mid x \text{ with } x \in \mathcal{X}$$

The introduction of the variables for the interaction with an external system is interesting insofar as it clearly separates the coordination aspect - what the MAS does - from the interaction with an external system aspect - the context in which the MAS is running. Thus, in the description $\{budget \leftarrow b, destination \leftarrow d\}$, regardless of which system is instantiating the variables b and d , the definition of the description and the behavior of the agent remain unchanged.

IV. SECURITY MANAGEMENT

We have decided to maintain global sharing of the data between all the agents, and not to isolate them in private environments, thus following the original Linda model. However, this choice leads to the same security problems. More precisely, fraudulent data insertion and retrieval could occur and the agents and the system designer cannot prevent them. In LACIOS, the agents are responsible of the objects that they put in the environment. In order to avoid fraudulent use of these objects, the language supports two control levels, a global level for the designer of the system to control the insertion of objects and a local level for the owner of the objects to control how their object will be used.

A. Global Control

The designer of the system knows the conditions under which certain insertions of objects are fraudulent and we provide him/her with a global control of agents insertions of objects. A threat to authenticity (when an agent tries to forge a message for example) is an example of such fraudulent insertions. More generally, objects added to the environment might corrupt the coherence of the data according to the application logic (resulting in two agents with the same *position*, or with a new bid that is lower than the current one, etc.).

Let us consider for instance, the following action:

$$add(\{from \leftarrow companion_2.id, to \leftarrow companion_1.id, subject \leftarrow \text{"coalition"}\})$$

This action is fraudulent, since the agent tries to send a message with a different *id* than its own. Therefore, this first class of threats concerns the security rules that have to be checked when an *add* is executed. To overcome threats resulting from the fraudulent adding of objects to the environment, the system

designer identifies the critical situations and specifies each one using a security rule s ($s \in \mathcal{S}, \mathcal{S} \subseteq Exp$ is the set of security rules of the system). An expression s in \mathcal{S} is a boolean expression in which the designer specifies the conditions on the state of the agent executing *add* and the conditions on the description of the object that it adds. To do so, we add a specific key word *that* in the syntax of an expression to designate, in a security rule, the object added by the agent.

$$e ::= \dots \mid \text{that}.e$$

For instance, here is the expression preventing an agent from adding an object that has a property *from* that is different from its own: $s \equiv \text{that}.from = id$, where *id* designates the identifier of the agent executing the *add* and *that* designates the object added by the agent. When an agent a executes $add(\{from \leftarrow companion_2.id, to \leftarrow companion_1.id, subject \leftarrow \text{"coalition"}\})$, the security rule specified by the designer is evaluated to false, because $d(from) \neq d_a(id)$, and the operation is canceled.

B. Local Control

The agents of the system know best the conditions under which the perception or retrieval of an object they add is fraudulent, and we provide them with local control to manage the observability of their own objects. A confidentiality threat (e.g. the interception by an agent of another's confidential information or message), or a threat to availability (e.g. the deletion of the agent's information or message by another agent) are examples of such fraudulent access. We propose to allow agents to define the observability rules - on perception and on retrieval - and to let the environment check that these conditions are respected.

This is done by enabling an agent, when it adds an object, to manage its observability, i.e. to identify the situations where the perception or retrieval of the added object is prohibited. To do so, the syntax of the primitive *add* is replaced as follows.

$$\mu ::= \dots \mid add(sds, e_p, e_r)$$

where e_p and e_r are boolean expressions. The expression e_p specifies the conditions that an agent has to satisfy to have the right to perceive the object described by *sds*, and e_r defines the conditions that an agent has to satisfy to have the right to retrieve it. When an agent executes $look(sds_p, sds_r, e)$, for each object $o \in C$ (the set of objects selected for matching from the environment) that is unified with a context variable in sds_p , the expression e_p associated to o has to be evaluated to true, and for each object o unified with a context variable in sds_r , the expression e_r associated with o has to be evaluated to true. Otherwise, the action *look* cannot be executed with this set of objects. When the agent doesn't want to restrain the perception or the retrieval of the object described by *sds*, it assigns true to e_p or e_r respectively. For instance, let agent a (let's say that a 's $companion.id = 5$) wants to prevent the message it has addressed to its *companion* to be retrieved by others, and to be perceived by any agent but itself (the key word *that* has the same semantics here, i.e. it designates the added

object): $add(\{from \leftarrow id, to \leftarrow companion.id, subject \leftarrow \text{“coalition”}\}, id = that.from, id = that.to)$

Consider an agent b with $d_b(id) = 10$ that executes $look(\{receiver \leftarrow r\}, \{message \leftarrow m\}, m.to = r.id \wedge r.destination = destination)$. The agent b is trying to retrieve a message (object unified with m) and to perceive the object representing the agent to which m is addressed (object unified with r), if its destination is equal to its own. Thanks to the conditions associated with the added object, b won't be able to perceive a 's message. Concretely, any matching that is trying to unify m with a 's message is prohibited by the environment and is not considered.

Note that, in the development of the security management defined above, we only take into account the security between local agents and the environment. By doing so, we make two assumptions. On the one side, the *spawn* of an agent representing an external system, user or agent, has to be fulfilled following a security protocol to ensure that this is indeed the agent with the claimed identifier. On the other side, we assume that local agents don't try to change their identifiers with an *update* throughout the execution of the system, which is easy to check before the execution of the system. Otherwise, they could dupe the global control mechanism.

V. SPECIFICATION OF AGENT BEHAVIOR

This section provides the complete definition of an open MAS written in LACIOS, starting with the complete definition of the primitives for LACIOS.

$$\mu ::= add(sds) \mid look(sds_p, sds_r, e) \mid update(sds) \mid spawn(P, sds)$$

We are now ready to define processes, which define agent behavior. The primitive $spawn(P, sds)$ launches a new agent that behaves like the process P and whose description is the result of the evaluation of sds (its transformation to a description ds). Below is the definition of a process, which defines agents' behaviors.

Definition 10 (Process): Given a set of process identifiers $\{K_i\}_{i \in I}$, a process definition is of the form: $\forall i \in I, K_i \stackrel{def}{=} P_i$, where every P_i is generated via the grammar in Table II.

$P ::= \mathbf{0}$	(null process)
$\mid \mu.P$	(action prefixing)
$\mid b[P] + b[P]$, where b is a boolean expression	(choice)
$\mid P \parallel P$	(parallel)
$\mid K_j$, for a certain $j \in I$	(invocation)
$\mid \nu X(P)$	(variables linking)
$\mu ::= spawn(P, sds) \mid add(sds) \mid look(sds_p, sds_r, e) \mid update(sds)$ with e an expression, sds, sds_p and sds_r symbolic descriptions	

TABLE II
PROCESS SYNTAX

Processes, ranged over by P, Q, \dots represent the programs of the MAS, and the behavior of its agents. A program can be a terminated process $\mathbf{0}$ (usually omitted). It can also be a choice expression between programs $b[P] + b[P]$, where each P is guarded by the evaluation of a boolean expression: when b is

evaluated to true, the program P is executed. A program can also be a parallel composition of programs $P \parallel Q$, i.e. P and Q are executed in parallel. A program can be an invocation of another process whose identifier is the constant K_j , and which behaves like the process defined by K_j . A program may be a process prefixed by an action $\mu.P$. Actions are the language primitives, as defined earlier. The operator ν is introduced to link free variables in P . The process $\nu X(P)$ introduces nondeterminism in the agents' behaviors. Indeed, behaves like P where every free variable (in X) is nondeterministically linked with a value in its type.

A coordinated MAS is then defined as follows.

Definition 11 (Coordinated MAS): $CS = \langle \Omega, d, \Omega_{ENV}, \mathcal{S} \rangle$

- $\Omega = \mathcal{A} \uplus \mathcal{O}$ is the set of entities, composed of \mathcal{A} the set of agents and \mathcal{O} the set of objects,
 - $\mathcal{A} \subseteq \Omega$ is the set of agents.
 - * Ω_a is the private memory of agent a , $\Omega_a \subseteq \mathcal{O} \cup \{a\}$, i.e. the agent has access to its own description,
 - * $proc(a)$ is the process defining the behavior of a .
 - $\mathcal{O} \subseteq \Omega$ is the set of objects.
 - * $e_p(o)$ returns the predicate specifying the perception conditions of o , i.e. which agents can perceive o .
 - * $e_r(o)$ returns the predicate specifying the retrieval conditions of o ,
- $d : \Omega \rightarrow (\mathcal{N} \rightarrow \mathcal{T})$ is the description function of the MAS, each $d(\omega)$ is an entity description as described before (denoted by d_ω as well),
- $\Omega_{ENV} \subseteq \mathcal{O}$ is the set of objects that are in the environment,
- $\mathcal{S} \subset Exp$ is the set of predicates specifying the conditions that have to be verified, in order for an *add* to be executed.

VI. THE PROGRAMMING LANGUAGE JAVA-LACIOS

We have defined a language that, following its operational semantics (cf. [13]), could be implemented in any host language. The usual procedure in order to implement a coordination language is to provide libraries in a host programming language that can be used by any system wishing to follow the coordination model (e.g. Klava, which is associated to Klaim [6]). However, to take full advantage of the language semantics, it is more useful not to require the programmer himself/herself to respect the semantics in each system that he/she implements. This is possible by providing him/her with a tool allowing to write a program in LACIOS's syntax, and to generate a system ready to be executed, with the guarantee that it respects the language semantics. In particular, we want to use the operators prefixing, choice and parallel composition when defining the agents' behaviors. Java has been chosen as a target programming language in which a compiled LACIOS program is translated, because of the relative simplicity of Thread management, as well as the easy creation of parsers thanks to the parser generator JavaCC ¹.

¹<http://javacc.dev.java.net/>

A JAVA-LACIOS program is a file where both the behaviors and the initial state of the coordinated MAS are described. A coordinated MAS is defined by the set of initial agents, spawned when the program starts, together with the security rules S . Programmers write their scripts which are parsed and compiled, generating a Java program. We have proposed a GUI for JAVA-LACIOS, which displays the ongoing execution, the current objects in the environment, the current agents' behaviors that are executed, etc. The Fig. 3 illustrates the execution with a Dial A Ride system that we have implemented [14]. It is also possible, before the execution, to visualize the graphs (labeled transition systems) related to the agents' behaviors.

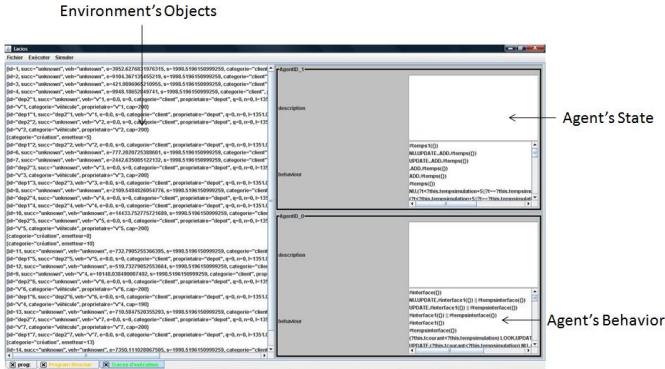


Fig. 3. Visualization of agents behaviors

The concurrent access to the environment objects with a *look* necessitates a synchronization of the *add* and *look* calls. However, an agent calling *add* has not to be blocked until the environment releases the lock. To this end, we define a buffer to which agents can add objects without blocking, while emptying the buffer is synchronized with the *look* calls.

when a *look* is called, the environment is locked while it is still looking for a matching, to guarantee that an agent does not access the environment in an incoherent state, and to be sure that a same object is not retrieved by more than one agent. If no matching is found, the calling process of the agent is blocked. The blocked processes are notified when an object is added to the environment. In this case, the notified process looks for a matching with the only newly added objects.

An *update* modifies the agent's properties *locally*, but it however influences its interaction with the environment. Indeed, if a *look* is currently executing, the matching have to be attempted with the current properties of the agent. When the properties of an agent change, and when they concern properties for which an ongoing *look* has attempted to match, the execution of the *look* is executed again, and the pending *look* requests are notified since they might be concerned by the newly changed properties as well.

VII. DISCUSSION AND RELATED WORK

Security is generally enforced by using multiple (logical) spaces, by stating "Interaction Laws" or by defining roles and access rights associated to them. With multiple spaces (e.g. Klaim [6], SecSpaces [8], [4] and SecOS [9]), two agents,

which wish to exchange confidential data use a space that is known only by the two of them. However, when security is guaranteed by isolating the data in private spaces, accessing one space gives the possibility to access all of its data, and being excluded from one space means not having access to any of its data. In LACIOS, agents have a state, and an agent can protect its data in a fine-grained way (at object level) without knowing the other agents, which allows secure interaction with complete data sharing. Roles and role access rights (like in the RBAC model associated to TuCSON [7]) are an additional layer on top of multiple spaces, and therefore security is also defined in a coarse-grained way.

In [8], specific cryptographic fields are added to the tuples to authenticate the producer of an item of data, for instance, as well as to identify the readers/takers of that item. This authentication is carried out in LACIOS thanks to agents' states and security rules, but it is nevertheless still possible to define a specific property for cryptographic fields.

The laws in LGI [10] allow data to be secured by specifying conditions on the states of the agents and the content of the data. Tuple-space reactions are associated with agents' actions so as to always result in a coherent configuration. Nevertheless, two points differentiate LACIOS from LGI. First laws in LGI are defined by the system designer only, whereas agents cannot do this. Second, laws are active rules, which poses the problem of non-termination of the matching process (action and a chain of endless reactions). In LACIOS, the rules cancel perceptions or retrievals but don't launch any reaction, so the problem of non-termination does not occur.

Tagged Sets [15] allow fine-grained protection of data added to the data space. However, neither agents' states nor powerful comparison operators are defined for it as in LACIOS.

In [5], the authors point out that "the secure version of Lime [[16]] is the only one which permits to control output operations, and SecSpaces [[8]] is the only one which permits to distinguish between the processes that can consume and the processes that can read a certain tuple". LACIOS allows for both insertions control and the distinction between reading a datum and taking it.

VIII. CONCLUSION AND PERSPECTIVES

The investigation of security issues in data driven coordination languages has lead us to propose a modified language allowing for fine-grained protection of exchanged data via the shared space. This paper has defined LACIOS, which can be used to model a large number of applications in which agents join and leave the system freely, where agents interact with external systems, and where security is crucial. Using LACIOS makes it easier for open MAS designers to translate the concepts manipulated by the agents and their interaction needs to LACIOS syntactic constructs, ensuring information security and expressing complex constraints. We have demonstrated this usefulness for a complex transportation application in our recent paper [14].

Our proposal is an entity oriented approach, and allows for the control of objects' insertion, perception and retrieval. It

distinguishes between objects' perception control and objects' retrieval control. The addition of agents' states, of *property-value* pairs data model, together with operators and variables lead us to propose a new language instead of building on top of an existing one. The formal operational semantics of LACIOS can be found in [13].

Our future works include the consideration of specific cryptographic properties to ensure authenticity. We are also investigating the addition of time constructs to LACIOS, inspired by the works of Busi *et al.* (e.g. [17]) and Linden *et al.* (e.g. [18]), to express temporary objects insertion and to define a deadline for *look* before termination with no effect. Interaction over multiple hosts is very challenging, yet with simple spaces, and with contextual interaction and the security mechanism, this becomes even more difficult to fulfill. Since we don't program mobile agents (as in Klaim [6] or Claim [19]), and since as a consequence the agents' locations are transparent at the language level, the concern of the environment distribution is to be tackled at the implementation level. Our ongoing research investigates the definition of architectures and strategies providing guidelines for environment distribution for LACIOS implementation.

REFERENCES

- [1] D. Gelernter, "Generative communication in linda," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 1, pp. 80–112, 1985.
- [2] P. Ciancarini, "Coordination languages for open systems design," in *Proceedings of the International Conference on Computer Languages (ICCL'90)*. New Orleans, LA (USA): IEEE Computer Society, 1990, pp. 252–260.
- [3] C. P. Pfleeger and S. L. Pfleeger, *Security in Computing*. Prentice Hall Professional Technical Reference, 2002.
- [4] M. Bravetti, N. Busi, R. Gorrieri, R. Lucchi, and G. Zavattaro, "Security issues in the tuple-space coordination model," in *Formal Aspects in Security and Trust*, T. Dimitrakos and F. Martinelli, Eds. Springer, 2004, pp. 1–12.
- [5] R. Focardi, R. Lucchi, and G. Zavattaro, "Secure shared data-space coordination languages: A process algebraic survey," *Sci. Comput. Program.*, vol. 63, no. 1, pp. 3–15, 2006.
- [6] R. De Nicola, G. L. Ferrari, and R. Pugliese, "Klaim: A Kernel Language for Agents Interaction and Mobility," *IEEE Transactions on Software Engineering*, vol. 24, no. 5, pp. 315–330, 1998.
- [7] A. Omicini, A. Ricci, and M. Viroli, "RBAC for organisation and security in an agent coordination infrastructure," *ENTCS*, vol. 128, no. 5, pp. 65–85, 2005, proceedings of the 2nd International Workshop on Security Issues in Coordination Models, Languages, and Systems (SecCo 2004).
- [8] N. Busi, R. Gorrieri, R. Lucchi, and G. Zavattaro, "Secspaces: a data-driven coordination model for environments open to untrusted agents," *Electr. Notes Theor. Comput. Sci.*, vol. 68, no. 3, 2003.
- [9] J. Vitek, C. Bryce, and M. Oriol, "Coordinating processes with secure spaces," *Sci. Comput. Program.*, vol. 46, no. 1-2, pp. 163–193, 2003.
- [10] N. H. Minsky, Y. Minsky, and V. Ungureanu, "Safe tuplespace-based coordination in multiagent systems," *Applied Artificial Intelligence*, vol. 15, no. 1, pp. 11–33, 2001.
- [11] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *Knowledge Engineering Review*, vol. 10, no. 2, pp. 115–152, 1995.
- [12] R. Milner, *Communication and Concurrency*. Prentice-Hall, 1989, 272 pages.
- [13] M. Zargayouna, "Coordination model and language for open multiagent systems. application to the dial-a-ride problem," PhD Dissertation, University of Paris-Dauphine, Paris (France), 2007, in french.
- [14] M. Zargayouna, F. Balbo, and G. Scmama, "A data-oriented coordination language for distributed transportation application," in *The third International KES Symposium on Agents and Multi-agent Systems Technologies and Applications (KES-AMSTA'09)*, ser. Lecture Notes in Artificial Intelligence. Uppsala (Sweden): Springer-Verlag, 2009, vol. 5559, pp. 283–292.
- [15] M. Oriol and M. Hicks, "Tagged sets: a secure and transparent coordination medium," in *Proceedings of the International Conference on Coordination Models and Languages (COORDINATION)*, ser. Lecture Notes in Computer Science, J.-M. Jacquet and G. P. Picco, Eds., vol. 3454. Springer-Verlag, April 2005, pp. 252–267.
- [16] A. L. Murphy, G. P. Picco, and G.-C. Roman, "LIME: A coordination model and middleware supporting mobility of hosts and agents," *ACM Trans. Softw. Eng. Methodol.*, vol. 15, no. 3, pp. 279–328, 2006.
- [17] N. Busi and G. Zavattaro, "Expired data collection in shared dataspace," *Theoretical Computer Science*, vol. 298, no. 3, pp. 529–556, 2003.
- [18] I. Linden, J.-M. Jacquet, K. D. Bosschere, and A. Brogi, "On the expressiveness of timed coordination models," *Science of Computer Programming*, vol. 61, no. 2, pp. 152–187, 2006.
- [19] A. Suna, "Claim & sympla : An environment for programming intelligent and mobile agents," PhD Dissertation, University of Paris VI, 2005, in french.

Executing Agent Plans by Reducing to Workflows

Tayfun Gokmen Halac, Övünç Çetin, Erdem Eser Ekinci
Rıza Cenk Erdur, Oguz Dikenelli

Ege University, Department Of Computer Engineering
35100 Bornova, Izmir, Turkey

Email: {tayfunhalac,ovuncetin,erdemeserekinci}@gmail.com
{cenk.erdur,oguz.dikenelli}@ege.edu.tr

Abstract—In this paper, we introduce an agent planner architecture that can reduce the basic artifacts of agent planning paradigms, semantic services and business process languages into a common workflow model. These artifacts are then executed by means of a workflow component that the architecture includes. By having a workflow component in an agent infrastructure, various agent programming paradigms including different planning approaches as well as different workflow definition languages can be executed on the same agent platform. To illustrate our ideas, we focus on the reduction of plans to the workflow model. To explicate the reduction mechanism, we have preferred to use HTN which is a widely known planning approach in multi-agent domain. Based on the semantics that we have defined for our workflow and HTN models, we have given an algorithm for transformation from HTN to workflow model.

I. INTRODUCTION

Agents can execute various task structures in order to achieve their goals. These task structures may be components of a plan (e.g. actions), services including semantically defined web services, or workflows which are represented using an appropriate formalism such as BPEL[1], XPDL[2]. An agent may execute each of these task structures in a way that is independent of others as it is the case for an agent that can execute only plans, only OWL-S service definitions or only workflows.

On the other hand, it is usually a desired property for an agent to execute several task structures in a combined manner. For example, one or more actions of a hierarchical task network (HTN)[3], [4] plan may need to call a web service or execute a pre-defined workflow. In addition, in open and collaborative multi-agent organizations where task structures can be distributed within the environment, it is required to discover, access, compose (if needed), and execute them at run-time. To support various task execution semantics both at design time and run-time, what is needed is a special agent planner architecture that should ideally provide a unique and common basis for the execution of different task structures in a both independent and combined manner.

There are three basic requirements to support various task execution semantics in an agent architecture. First, meta-models for the representation of various task semantics are needed. OWL-S, which is a standard for defining web services semantically, and workflow definition formalisms such as BPEL are examples for such meta-models. As another example, agent plans can be represented using OWL ontologies at

the meta-level. Second, a common model that will form a common execution basis for the tasks that have different semantics is needed. Based on the fact that a plan can be represented as a directed graph which can be executed as a workflow, defining a generic workflow graph model will satisfy the requirement for a common model. Finally, algorithms for the transformations from the meta-models into the common representation model should be developed.

In this paper, we introduce a planner architecture that fulfills the requirements given above. The introduced architecture includes a generic workflow graph model into which various task semantics can be transformed. This generic workflow graph model has been defined based on the abstract definition given in [5]. Within the planner architecture, we have also implemented an executor component which is used to execute the instances of the generic workflow graph model.

In literature, there are studies that aim to execute web services or workflows within a planner or an agent architecture. [6] describes how SHOP2 HTN planning system can be used to execute OWL-S descriptions. The SHOP2 planner takes the composite process defined using OWL-S as input and executes this composite process. WADE[7] is a software platform which is built on top of the well-known agent framework JADE[8]. WADE uses a directly executable simple workflow structure based on java class instead semantically described planning paradigms. Our study differs from these works, since our proposed planner architecture can support combinations of various task semantics both at design-time and run-time by providing a common execution layer for all of them. Neither [6] nor [7] aims to support more than one task execution semantics at the same time. Another point that needs attention is that the workflow graph model which constitutes the core of our common execution model is not related with the concept of executing a workflow within an agent. The workflow graph model is a directed graph structure into which various task semantics are transformed before execution.

We have implemented the planner architecture within SEAGENT[9], which is a semantic web enabled multi-agent system framework developed by our research group[10]. The planner can reduce the plans defined using OWL ontologies and OWL-S service definitions into the common workflow graph model, and then execute them. To illustrate the reduction process, we just focus on the transformation of HTN semantics into the common workflow graph model in this paper. We have chosen HTN because HTN planning is a well-known approach

that has affected the agent domain most, and has been directly used in several agent development frameworks[11], [12]. SEAGENT also incorporates HTN as its planning paradigm.

Remaining parts are organized as follows: Before giving the details of the reduction mechanism, we introduce current architecture of SEAGENT planner in Section II. In section III, details of our planner's internal workflow structure, to which HTN plans and other process definition languages are reduced, are given. Soon after, we define our enhanced HTN semantics in section IV. In section V, the algorithm that achieves the reduction from HTN to Workflow model is given, and correctness of the reduction mechanism is discussed. Section VI includes a working example and Section VII the conclusion.

II. THE ARCHITECTURE OF THE PLANNER

The implementation of the proposed planner architecture is presented in Figure-1. As indicated in the figure, two vertical layers compose the overall architecture. The front layer, called Semantic Web Layer, uses ontological descriptions to represent the artifacts of the planning process. These ontology descriptions are: *Goal*, *Role*, *HTN*¹ and *OWL-S*. The goal ontology defines the objectives which the agent intends to reach within the organization. It specifies abstract definitions for agent behaviors. The role ontology, on the other hand, puts the related goal definitions together within a role description. In addition, it also defines some constraints about the considered role. Roles and goals take in charge during planning process, and then the output plan is reduced to the workflow model. Our main interest is on reduction of plans, not on this plan decision process. The HTN ontology is used to describe the agent plans using HTN planning technique. Finally, OWL-S (Web Ontology Language for Services) is a part of our proposed architecture to support semantic services. As is seen in the figure, the planner, which composes the Execution Layer, consists of four modules: *WorkflowElements*, *WorkflowEngine*, *Matchers*, *Reducers*.

The *WorkflowElements* module contains building blocks of our graph structure which is used to represent the agent acts as a workflow at execution time. The graph structure is based on tasks, coordinators and flows, all of which are discussed in detail in Section III.

The *WorkflowEngine* module is responsible for performing a workflow instance and coordinating its execution. It consists of three submodules: *ExecutionToken*, *LazyBinder*, and *GraphVerifier*. The *ExecutionToken* is the major component for the execution which traverses the nodes of the workflow instance and executes tasks of workflow instance. The *LazyBinder* module was developed to support dynamic graph binding. It has a special abstract graph node called *LazyTask* which loads the concrete graph at runtime. This dynamic binding capability makes our workflow dynamically extendable. Thus, at run-time new goals and tasks can be appended based on the state of the environment and/or agent's knowledge. Finally, the *GraphVerifier* module is responsible for verification of a workflow instance before it is given to the

execution. It verifies the syntactical structure of the workflow and data flow over this workflow instance.

Reducers, Matchers, and *JENA*² form a bridge connecting the execution layer to the Semantic Web layer. The ontological definitions of the agent's actions from the Semantic Web layer are read here, and converted to the corresponding workflows. The Reducers sub-package contains the graph reducer components (*GoalReducer*, *HTNReducer*, *OWLSReducer*) that parse Goal, HTN and OWL-S definitions and reduce them into the workflow. The other module, called Matchers, contains three submodules: *RoleMatcher*, *GoalMatcher* and *ActMatcher*. Role and goal matchers collaborate to find an appropriate goal description for an objective which the agent must fulfill. If a goal is matched with an objective, the description of the goal is given to the *GoalReducer* to reduce it into a goal workflow. During the execution of the goal workflow, an *ActMatcher* is invoked for each sub-goal to find an appropriate act (HTN plan or OWLS service) that accomplishes the sub-goal. After the *ActMatcher* returns the corresponding act description, a reducer (*HTNReducer*, or *OWLSReducer*) is selected to reduce the act description to the workflow.

In this paper, we only focus on the reduction of HTN plans to workflows. To specify our reducing process, workflow and HTN semantics are formally defined in the following sections. Next, reducer algorithm is illustrated in the section V.

III. SEMANTICS OF SEAGENT WORKFLOW GRAPH

As mentioned in the introduction, workflow technology has been extensively researched in the academy and as a result of these efforts this technology has reached to a high degree of maturity. Widely acceptance of workflows in industrial settings and standardization of workflow definition languages such as BPEL, XPD, are the signs of this maturity degree. On the other hand, from the execution perspective, several approaches have raised up such as executing the workflows on Petri-Nets[13] and on conceptual directed graphs[5].

Sadiq and Orłowska abstractly introduced the basic building blocks of a workflow process using the graphical terms such as nodes and flows in [5]. They also defined the basic workflow constructs such as sequence, choice, concurrency and iteration. Besides the modeling workflows, they touch on reliability of the workflow model in [14], [15]. For this purpose, some structural conflicts as deadlock, lack of synchronization, live-lock are determined. Our workflow model is built by deriving the abstract definitions of Sadiq et al. and extended by constraints to avoid structural conflicts articulated in [14].

In this section, we semantically declare the concepts and their constraints of our workflow implementation. Before giving semantics of our workflow model, we evoke to the reader that a workflow is also a kind of directed graph[16], [17]. So, we start to explain semantics of our model by giving the formalism of the directed graph with the following definition.

Definition 1: Given directed graph is a tuple of $g = \langle V, E \rangle$, $V = \{v_0, v_1, \dots, v_n\}$ and $E = \{\langle v_s, v_t \rangle : v_s, v_t \in V\}$.

²We use JENA (<http://jena.sourceforge.net/>) to read and manipulate the ontology documents in the knowledge-base and over the Internet.

¹<http://etmen.ege.edu.tr/etmen/ontologies/HTNOntology-2.1.4.owl>

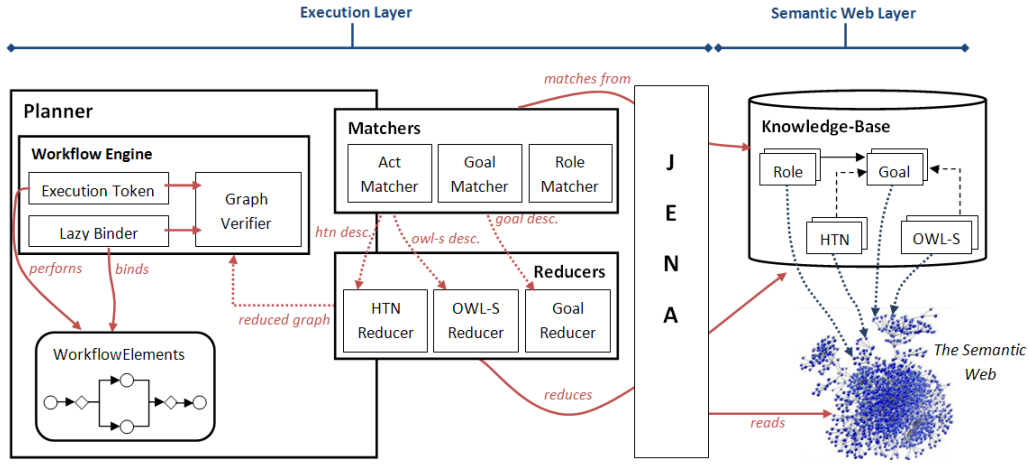


Figure 1. SEAGENT Planner Architecture

The directed graph represented with g consists of vertices V and directed edges E . A vertex, $v \in V$, specifies a node in the graph. The words, *node* and *vertex*, will be used interchangeably. A directed edge, $e \in E$, shows a directed link between two vertices of the graph. In the definition, vertex " v_s " represents the source vertex of the edge and vertex " v_t " is for the target. We define a function, *path*, that helps to gather the directed ways between two given vertices:

- $path(v_i, v_k) = (e_0, \dots, e_n)$ where $v_i \in V$ and $v_k \in V$ represents the first and the last vertex of the path. *path* defines one of the directed ways between vertices v_i and v_k . The first term of the finite sequence of edges is e_0 where $source(e_0) = v_i$ and the last term is e_n where $target(e_n) = v_k$. For all terms of the sequence, target node of an edge equals to source node of the next term, $(target(e_0) = source(e_1)) \wedge \dots \wedge (target(e_{n-1}) = source(e_n))$.
- $paths(v_i, v_k) = \{path_1(v_i, v_k), path_2(v_i, v_k), \dots\}$ represents all different ways between the given two nodes.

This definition uses two functions for each edge $e \in E$: $source(e) = v_m$ where $v_m \in V$ represents the source vertex of e and $target(e) = v_n$ where $v_n \in V$ represents the target vertex of e .

Semantics of the workflow graph model, which extends the directed graph represented formally above, is defined below by giving details of the model's building blocks.

Definition 2: *wfg*, which is a tuple $\langle T, C, CF, DF, IC_{wfg}, OC_{wfg}, TN \rangle$, expresses a *workflow graph* that consists of set of tasks T , set of flows CF and DF which represent control flow and data flow sets respectively, input and output containers IC_{wfg} and OC_{wfg} , set of coordinators C , and set of terminal nodes TN .

The workflow graph as mentioned previously is derived from the directed graph. So, when looked through the directed graph perspective, some entities of workflow graph, such as tasks, coordinators and terminal nodes, are sub-sets of the vertex set: $T, C, TN \subset V$. Also, CF and DF are specialized entities of workflow that are sub-sets of directed edge set: $CF, DF \subset E$.

Definition 3: An element of *task* set is formally defined as

$\tau_i = \langle n_{\tau_i}, IC_{\tau_i}, OC_{\tau_i} \rangle \in T$. In detail, n_{τ_i} is the identifier of the task, while IC_{τ_i} and OC_{τ_i} correspond to input and output containers respectively.

- A data container, κ , is the set of data items which are needed to execute a task or generated by the task, $\kappa = \{d_1, d_2, \dots, d_i\}$. IC and OC are sub types of data container, $IC, OC \subset \kappa$.
- A data item, $d_i = \langle n_{d_i}, type_{d_i} \rangle$, stands for data which is held by input and output containers. d_i is identified by its name (n_{d_i}) within the container and $type_{d_i}$ property specifies the data-type of d_i .

The required coordination of tasks of workflow and data sharing between tasks are provided by special components called *flows*. There are two types of flows in our model: *control flows* and *data flows*. The details of the flows are determined with the following definitions.

Definition 4: A *data flow*, $df_i = \langle \tau_{src}, d_{src}, \tau_{trg}, d_{trg} \rangle \in DF$ where $\tau_{src}, \tau_{trg} \in T$, $d_{src} \in OC_{\tau_{src}}$, $d_{trg} \in IC_{\tau_{trg}}$, is a type of flow that is used for data migration between two tasks. It transmits the value of the source task (τ_{src})'s output item (d_{src}) to the target task (τ_{trg})'s input item (d_{trg}) after the source task performed.

Data flows are important to supply inputs to the tasks. So, to supply the inputs safely, we define some constraints on data flows. Before declaring these constraints, we define two supportive functions, *inData* and *outData*, as below:

$outData(d_{src}) = \{df : df \in DF \wedge sourceItem(df) = d_{src}\}$ where d_{src} is a data item, returns the set of data flows whose source data item is d_{src} .

$inData(d_{trg}) = \{df : df \in DF \wedge targetItem(df) = d_{trg}\}$ where d_{trg} is a data item. It returns the set of data flows whose target is d_{trg} .

sourceItem and *targetItem* functions are similar to *source* and *target* functions, but they are used to retrieve the data items bound by the specified data flow.

Now, we can describe the constraints on data flows using these functions:

- **(C)** There should not be more than one data flow between any two data items: $\forall d_{src}, d_{trg} (|outData(d_{src}) \cap inData(d_{trg})| \leq 1)$

- **(C)** Data type of the target task's input item must be equal or subsume the type of the source task's output item: $\forall df \in DF \ (type_{d_{src}} \subseteq type_{d_{trg}})$

As we mentioned above, a data flow expresses the direction of the data item migration. Differently from the data flow, on the other hand, a control flow is used to specify the execution sequence of task nodes in a workflow graph.

Definition 5: A control flow is a tuple, $cf_i = \langle v_{src}, v_{trg} \rangle \in CF$, consisting of source vertex (v_{src}) and target vertex (v_{trg}). Control flows are more decisive than data flows on process of workflows. To avoid inconsistencies, control flows must be constructed according to some defined constraints. Before declaring these constraints on our workflow model, we describe two supportive functions, *inControl* and *outControl*, to make the constraints more understandable:

$inControl(n) = \{cf : cf \in CF \wedge target(cf) = n\}$ where $n \in V$, acquires the set of control flows whose target node is n .
 $outControl(n) = \{cf : cf \in CF \wedge source(cf) = n\}$ where $n \in V$, returns the set of control flows whose source is n .

Now, we can specify the constraints using these functions:

- **(C)** All flows must have two different nodes connected to their two sides. $\forall f \in E \ (source(f) \neq target(f))$
- **(C)** A task have to be source or target of only one control flow. $\forall \tau \in T \ (|inControl(\tau)| = 1 \wedge |outControl(\tau)| = 1 \wedge inControl(\tau) \neq outControl(\tau))$
- **(C)** There should not be more than one direct control flow between any two nodes. $\forall v_m \in V, \forall v_n \in V \ (|outControl(v_n) \cap inControl(v_m)| \neq 1)$
- **(C)** The source node of a control flow must be ahead of the target node in the order of execution. $\forall v_m, \forall v_n \ ((path(v_m, v_n) \neq \emptyset) \rightarrow (outControl(v_n) \cap inControl(v_m) = \emptyset))$. As an exception, in an iteration structure, one of the outgoing flows of a choice node goes to the preceding merge node.

Although the constraints on control and data flows help to build consistent work flows, they are not enough. Control and data flows also must be compatible with each other in terms of workflow direction.

- **(C)** Since a data flow transmits the data from the source node to the target node, the source node must be finished before the target node starts. Therefore, the source node must always precede the target node within the workflow in terms of execution sequence.

$\forall \tau_m, \forall \tau_n \ ((path(\tau_m, \tau_n) \neq \emptyset) \rightarrow outData(\tau_m) \cap inData(\tau_n) = \emptyset)$

$\forall \tau_m, \forall \tau_n \ ((outData(\tau_m) \cap inData(\tau_n) \neq \emptyset) \rightarrow (path(\tau_m, \tau_n) \neq \emptyset))$

$inData(\tau) = \{df : df \in DF \wedge target(df) = \tau\}$ where $\tau \in T$, returns the set of data flows whose target is τ .

$outData(\tau) = \{df : df \in DF \wedge source(df) = \tau\}$ where $\tau \in T$, returns the set of data flows whose source is the given task.

Data migration on the workflow and execution ordering of the tasks can be provided easily via flows. But they are not sufficient when some complex structures, which come from the nature of the workflow, such as concurrency, alternation and iteration, are considered. We need some special nodes for

the purpose of implementing these structures. These special nodes named as *coordinator nodes*, will be defined next.

Definition 6: Here, we define all sub-sets of *coordinators*, C , together. There are four types of coordinator nodes; *choice*, *merge*, *fork*, *synchronizer*; $CH, MR, FR, SY \subset C \subset V$.

- A *choice* node, $ch_i = \langle n_{ch}, d_{cond} \rangle \in CH$, has more than one outgoing flows and contains a condition input d_{cond} to select a branch.
- A *merge* node, $mr_i \in MR$, has more than one incoming branches and it is used to join the mutually exclusive paths which are split by a choice node.
- A *fork* node, $fr_i \in FR$, has more than one outgoing flows and enables all of them at the same time.
- A *synchronizer* node, $sy_i \in SY$, has more than one incoming flows, which are activated concurrently by a fork node, and waits all to be finished. In other words, it synchronizes the concurrent execution paths within a workflow.

The coordinator nodes are used in pairs to construct exclusive choice and parallel split workflow patterns[18]. The exclusive choice pattern creates a divergence point into two or more branches such that only one of which is selected for the execution. The parallel split pattern, on the other hand, provides concurrent execution paths which are activated simultaneously. An exclusive choice is constructed with a $\langle ch_i, mr_i \rangle$ pair, while a $\langle fr_i, sy_i \rangle$ pair composes a parallel split.

As is clearly stated, the coordinator nodes are required to build workflows including complex patterns. But misuseage of the coordinators may result in defected workflows. Therefore, the following constraints should be defined on the coordinator nodes to provide a consistent workflow.

- **(C)** Since the aforementioned workflow patterns are constructed using coordinator pairs, there must exist a merge node for each choice node, and a synchronizer node for each fork node. These two constraints could be expressed by $f : CH \rightarrow MR$ and $g : FR \rightarrow SY$ functions, which are one-to-one and onto, respectively.
- **(C)** All choice and fork nodes have one incoming and more than one outgoing flows. $\forall n \in FR \cup CH \ ((|inControl(n)| = 1) \wedge (|outControl(n)| > 1))$
- **(C)** All synchronizer and merge nodes have at least two incoming flows and only one outgoing flow. $\forall n \in SY \cup MR \ ((|inControl(n)| > 1) \wedge (|outControl(n)| = 1))$

The node definitions made so far specify the intermediate nodes. In other words, we did not give any definition of nodes which represents the end points of the workflow up to now. The following definition, on the other hand, explains the terminal nodes, $TN = \{v_i, v_f\}$, used for this purpose.

Definition 7: *Initial node*, $v_i \in V$, represents the beginning of the workflow, while the *final node*, $v_f \in V$, represents the end, $wfg_n = (v_i, \{\tau_1, \tau_2, \dots, \tau_n\}, \{c_1, c_2, \dots, c_n\}, v_f)$.

- **(C)** v_i is the first node of the wfg_n , it has no incoming but only one outgoing control flow: $\forall v_i \ ((inControl(v_i) = \emptyset) \wedge (|outControl(v_i)| = 1))$
- **(C)** v_f is the last node of the wfg_n , it has only one incoming but no outgoing control flow: $\forall v_f \ ((inControl(v_f) = 1) \wedge (outControl(v_f) = \emptyset))$

- **(C)** Each workflow graph contains exactly one initial and one final node.

$$\forall wfg (wfg \in WFG \rightarrow wfg \ni v_i \wedge wfg \ni v_f)$$

IV. SEMANTICS OF SEAGENT HTN

Previously, we gave semantics of our workflow model. Our approach, as mentioned in the introduction, is to design and implement a planner architecture that enables to execute different planning paradigms and workflow definition languages in the same agent architecture. Due to this purpose, we choose HTN paradigm, mostly used planning paradigm in the agent literature.

Semantics of HTN is firstly articulated by Kutluhan et al. in [3]. In his model, HTN is closer to AI problem solving. For the purpose of using HTN in web agent programming, Sycara et al. reformed it in [4]. They contributed the link concept to provide a unified information and control flow within plans. Although that contribution makes HTN plans more tractable, it allows designing error-prone plans. Our HTN model is a detailed revision of Sycara et al.'s that is extended by exhaustive link definitions and constraints that permit to avoid designing erroneous plans. Base concept of our HTN model is *task*;

Definition 8: An HTN task, $\theta_i = \langle n_{\theta_i}, P_{\theta_i}, O_{\theta_i} \rangle \in \Theta$, is generalization of the primitive task (action) and the compound task (behavior), $A \subset \Theta$, $B \subset \Theta$.

A task encapsulates the common properties of behaviors and actions, such as provisions, outcomes, and name. But they are distinguished by other properties explained below.

Definition 9: A behavior, $\beta_i = \langle n_{\beta_i}, P_{\beta_i}, O_{\beta_i}, ST_{\beta_i}, \Gamma_{\beta_i} \rangle \in B$, represents a compound task which encloses other tasks. A behavior (β_i) corresponds to a workflow graph (*wfg*) that has sub nodes which may be primitive actions or other behaviors. In the definition, a behavior consists of name, provisions, outcomes, subtasks, and subtask links respectively. n_{β_i} is a label that distinguishes the behavior from the others. Since a behavior is a compound task, it cannot be performed directly. It must be decomposed to its primitive actions whose executions contribute toward accomplishment of the behavior.

Definition 10: An action, $\alpha_i = \langle n_{\alpha_i}, P_{\alpha_i}, O_{\alpha_i}, \lambda_{\alpha_i} \rangle \in A$, is a primitive task that is executed directly. It corresponds an indecomposable workflow task node.

An action consists of name, provisions, outcomes, and a function. n_{α_i} is a label that distinguishes the action from the others. Because actions are executable entities within a plan, they must implement a function, λ_{α_i} , which fulfills the actual work of the action.

Definition 11: A parameter, $\pi_i = \langle n_{\pi_i}, type_{\pi_i} \rangle \in \Pi$, stands for data which is needed or produced by tasks. A parameter π_i consists of name (n_{π_i}) and type ($type_{\pi_i}$).

The parameter is an abstract concept that cannot be seen in an HTN plan. Parameter is generalization of provision and outcome, $P \subset \Pi$, $O \subset \Pi$. The definitions of these concepts are given below.

Definition 12: A provision, $p_i = \langle n_{p_i}, type_{p_i}, value_{p_i} \rangle \in P$, represents the data which is needed for execution of task.

Within a workflow, P corresponds an input container and each data item in it represents a provision. Therefore, it provides the data which is required for starting the task's execution and whose value can be obtained from an outcome of the preceding task or from an external resource.

Definition 13: An outcome, $o_i = \langle n_{o_i}, type_{o_i}, value_{o_i} \rangle \in O$, represents the data which is produced during execution. Some tasks gather information, which is represented by outcomes, during execution. They can be passed to needer tasks. O corresponds the output container, which consists of data items that represents outcomes, of a task within a workflow.

Definition 14: A state (or outcome state), $s_i \in S$, is a label on a link specifying that the branch will be executed in which condition.

State instances are used to construct branching or concurrency structures within plans. In detail, outgoing links with distinct outcome state results in an exclusive choice pattern, while the same outcome states form a parallel split.

Definition 15: $ST = \{\theta_1, \theta_2, \dots, \theta_i\}$ indicates the set of *subtasks* of a behavior.

A constraint on subtasks is revealed below.

- **(C)** A task can be child of exactly one behavior (except the root behavior which represents the HTN plan). In other words, a task can be included by only one ST . $\forall \theta \in \Theta (\theta \in ST_{\beta_m} \rightarrow \theta \notin ST_{\beta_n})$

Up till now, we have mentioned about the HTN actions, behaviors and relation between a behavior and its subtasks. For the rest of this section, link definitions, which forms control and information flows between tasks, will be given. For that purpose, we define the link set, Γ , that is the super set of provision, order, inheritance and disinheritance links, $PL, OL, IL, DL \subset \Gamma$.

- **(C)** An important constraint on links: The source and the target task of a link must be different: $\forall l \in \Gamma (source(l) \neq target(l))$

Here we define two functions that are used in determining the details of links: For $\forall \theta \in \Theta$

$$\begin{aligned} inLink(\theta) &= \{link : (link \in \Gamma) \wedge (target(link) = \theta)\} \\ outLink(\theta) &= \{link : (link \in \Gamma) \wedge (source(link) = \theta)\} \end{aligned}$$

As mentioned, there are four types of link: *provision link*, *order link*, *inheritance link* and *disinheritance link*. While provision links coincide with both data and control flows, order links correspond to control flows only. Inheritance and disinheritance links are data flows between a behavior and its subtasks. Here, the formal definitions of links and their constraints are given with necessary logical functions.

Definition 16: An order link, $oLink_i = \langle \theta_{src}, \theta_{trg}, s \rangle \in OL$, represents a control flow between two tasks and it designates the execution order.

Order links consist of source task, target task, and state. By using order links and states together, we can create plans including conditional branching and parallel execution paths.

- **(C)** Source and target tasks of an order link must be included in the same subtask list. In other words, an order link can connect two tasks if both are the subtask of the same behavior. $\forall link \in OL (source(link) \in ST_{\beta_n} \leftrightarrow target(link) \in ST_{\beta_n})$

- **(C)** At most one order link can be constructed between two tasks. $\forall \theta_{src}, \forall \theta_{trg} (|(outLink(\theta_{src}) \in OL) \cap (inLink(\theta_{trg}) \in OL)| \leq 1)$

We define a generalized concept, *parameter link* ($\pi\mathbb{L}$), where $PL, DL, IL \subset \Pi\mathbb{L}$, for the rest of link types: *inheritance*, *disinheritance* and *provision links*. All these links constructs data flows between the tasks by mapping the source and the target parameter of these tasks. A significant point about the parameter mapping is compatibility of the parameter types: $\forall \pi\mathbb{L} \in \Pi\mathbb{L} (type(sourceParam(\pi\mathbb{L})) \subseteq type(targetParam(\pi\mathbb{L})))$ where *sourceParam* and *targetParam* functions are used to retrieve the source and the target parameter of the parameter link respectively.

Two definitions below specify two supportive functions that are used to get incoming and outgoing parameter links which are used to bind the given parameter. These functions will help to describe the later definitions.

$inLink(\pi) = \{\pi\mathbb{L} : (\pi\mathbb{L} \in \Pi\mathbb{L}) \wedge (targetParam(\pi\mathbb{L}) = \pi)\}$ where $\forall \pi \in \Pi$.

$outLink(\pi) = \{\pi\mathbb{L} : (\pi\mathbb{L} \in \Pi\mathbb{L}) \wedge (sourceParam(\pi\mathbb{L}) = \pi)\}$ where $\forall \pi \in \Pi$.

Definition 17: A *provision link*, $pLink_i = \langle \theta_{src}, \theta_{trg}, o_s, p_t, s \rangle \in PL$, represents a data and a control flow between two tasks.

A provision link binds an outcome of the source task and a provision of the target task. If a conditional branching occurs after the source task, the outcome state of the link (s) maps a particular branching condition to the target task.

- **(C)** Source and target tasks of a provision link must be the child of the same behavior. $\forall pLink \in PL (source(pLink) \in ST_{\beta_n} \leftrightarrow target(pLink) \in ST_{\beta_n})$
- **(C)** Source parameter of a provision link must be an outcome and target parameter must be a provision. $\forall pl \in PL ((sourceParam(pl) \in O) \wedge (targetParam(pl) \in P))$
- **(C)** At most one provision link can be constructed between the same outcome-provision pair. $\forall o_{src}, \forall p_{trg} (|(outLink(o_{src}) \cap PL) \cap (inLink(p_{trg}) \cap PL)| \leq 1)$
- **(C)** Either an order link or a provision link can be constructed between two tasks.

$\forall \theta_{src}, \forall \theta_{trg} ((outLink(\theta_{src}) \cap inLink(\theta_{trg}) \cap OL \neq \emptyset) \rightarrow (outLink(\theta_{src}) \cap inLink(\theta_{trg}) \cap PL = \emptyset))$

$\forall \theta_{src}, \forall \theta_{trg} ((outLink(\theta_{src}) \cap inLink(\theta_{trg}) \cap PL \neq \emptyset) \rightarrow (outLink(\theta_{src}) \cap inLink(\theta_{trg}) \cap OL = \emptyset))$

Definition 18: An *inheritance link*, $iLink_i = \langle \beta_{src}, \theta_{trg}, p_s, p_t \rangle \in IL$, represents a parameter link between a behavior and one of its subtasks. It corresponds to a data flow from the initial node of a workflow to a subtask.

Inheritance link consists of source behavior, target task, a provision of source behavior, and a provision of target sub task. $\beta_{src} \in B$ and $\theta_{trg} \in ST_{\beta_{src}}$.

- **(C)** Source and target parameter of an inheritance link must be a provision. $\forall il \in IL ((sourceParam(il) \in P) \wedge (targetParam(il) \in P))$
- **(C)** At most one inheritance link can be constructed between the same provision pairs. $\forall p_{src}, \forall p_{trg} (|outLink(p_{src}) \cap inLink(p_{trg}) \cap IL| \leq 1)$

- **(C)** Each provision of the root behavior must be bound with at least one inheritance link. $\forall p \in P_{\beta_{root}} (|(outLink(p) \cap IL)| > 0)$ where $\beta_{root} \in B$.

Definition 19: A *disinheritance link*, $dLink_i = \langle \theta_{src}, \beta_{trg}, o_s, o_t \rangle \in DL$, represents a parameter transfer from a task to parent behavior. It corresponds to a data flow from a subtask to the final node of a workflow.

Disinheritance link consists of source task, target behavior, an outcome of source sub task, and an outcome of target behavior. Source task of a disinheritance is child of the target task, $\beta_{trg} \in B$ and $\theta_{src} \in ST_{\beta_{trg}}$.

- **(C)** Source and target parameter of a disinheritance link must be an outcome. $\forall dl \in DL ((sourceParam(dl) \in O) \wedge (targetParam(dl) \in O))$
- **(C)** At most one disinheritance link can be constructed between the same outcome pairs. $\forall o_{src}, \forall o_{trg} (|outLink(o_{src}) \cap inLink(o_{trg}) \cap DL| \leq 1)$
- **(C)** A disinheritance link must be provided for each outcome of a behavior to collect all outcomes from the sub-tasks. If there is an exclusive choice structure, a disinheritance link must be constructed for all exclusive paths to fulfill all outcomes of the behavior. $\forall \beta_i \in B (\forall o_n \in O_{\beta_i} (|inLink(o_n) \cap DL| > 1))$

V. TRANSFORMATION OF HTN INTO WORKFLOW

To implement our approach about executing different planning paradigms and workflow definition languages in the same agent architecture, *HTNReducer*, which is a component of the Reducers submodule as mentioned in Section II, is used to transform an HTN definition into a workflow before execution. In this section, this transformation process is introduced within the scope of our workflow-based HTN planner.

Algorithm 1 Reduction of an HTN behavior to a workflow

Input: an HTN behavior β .

Output: a workflow wfg .

- 1) Initiate a workflow graph wfg corresponding to β .
 - 2) Create the nodes corresponding to the subtasks of β .
 - a) If subtask is an HTN behavior, then apply the same process from step 1 and create a complete subworkflow for the subbehavior.
 - b) For an HTN action, otherwise, create a primitive workflow task node.
 - 3) Construct flows between workflow tasks in wfg .
 - 4) Put required coordinator nodes to junction points.
-

A. Reduction Algorithm

Based on the formal definitions in Section III and Section IV, we have developed an algorithm shown in Algorithm-1 for reducing a behavior description to the corresponding workflow. For this purpose, the HTN reduction algorithm constructs a part of the graph in a few steps. It begins the process with creating an empty graph, which consists of initial and final nodes, and the data containers only, for the behavior. Then, it creates workflow task nodes for the subtasks of the behavior

and adds them into the empty graph. After the subtask nodes are added to the graph model, the next step is constructing the flows between these nodes. Finally, the last step of the reduction process, which follows the flow construction, is placing the coordinator nodes to the required locations within the graph. The steps of the algorithm are elaborated below.

In step 2, a node is created for each subtask of the given behavior according to its type. If the subtask is an HTN action, a primitive task node is constructed together its data containers. Otherwise, for an HTN behavior, the reduction process is achieved for this behavior and a graph model is created. The point to take into consideration is recursion while creating subnodes of workflow. By means of this recursion a sub-behavior is reduced prior to its parent.

As previously mentioned, the next step following the subtask creation is connecting the flows between these nodes. To do this, appropriate flow(s) for each link that is defined by the behavior description is constructed. The link type specifies the flow type and end nodes of the flow. For an inheritance link, a data flow from the initial node of the graph (i_{wfg}) to the target task of the link is constructed. A disinheritance link corresponds to a data flow between the source task of the link and the final node of the graph (f_{wfg}). For order links and provision links, on the other hand, a control flow is constructed for each. In addition to the control flow, a data flow is also constructed for a provision link.

After the flow construction phase, we obtain a raw graph model that consists of only task nodes and flows between them. There is no coordination component in this model. The last step of the algorithm, in line 4, overcomes this lack by placing the coordinators to the appropriate locations within the graph. To do this, the divergence and convergence points are marked with special nodes and then these special nodes are replaced with suitable coordinator nodes.

As a result, at the end of the reduction process, we obtain a complete workflow graph corresponding to the given HTN behavior. The graph contains task nodes that are connected with the flow objects, and the coordinator nodes that determine the flow of execution.

The outputs of our algorithm for primitive HTN patterns are represented in Figure-2. These patterns are composed of behaviors which have only primitive subtasks and other building blocks of HTN. Since plans, which have only primitive subtasks, can be defined by assembling these patterns, adding new actions to them and constructing new links between actions, they can be transformed into workflows.

To understand the reduction process better, we explain it by demonstrating one of primitive patterns. (see Figure-2(F)) The input behavior can be represented as $\beta_1 = \langle \langle BH1' \rangle, \{p_{\beta_1}\}, \{o_{\beta_1}\}, \{\alpha_1, \alpha_2, \alpha_3\}, \{pLink_1, pLink_2\} \rangle$ where $\alpha_1 = \langle \langle AC1' \rangle, \{p_{\alpha_1}\}, \{o_{\alpha_1}\}, \lambda_{\alpha_1} \rangle$, $\alpha_2 = \langle \langle AC2' \rangle, \{p_{\alpha_2}\}, \{o_{\alpha_2}\}, \lambda_{\alpha_2} \rangle$, $\alpha_3 = \langle \langle AC3' \rangle, \{p_{\alpha_3}\}, \{o_{\alpha_3}\}, \lambda_{\alpha_3} \rangle$ and $pLink_1 = \langle \alpha_1, \alpha_2, o_{\alpha_1}, p_{\alpha_2}, S'_1 \rangle$, $pLink_2 = \langle \alpha_1, \alpha_3, o_{\alpha_1}, p_{\alpha_3}, S'_1 \rangle$.

- At the start, an empty workflow $w_{f_1} = \langle \emptyset, \emptyset, \{i_{w_{f_1}}, f_{w_{f_1}}\}, \emptyset, \emptyset, \emptyset, \{i_{w_{f_1}}, f_{w_{f_1}}\} \rangle$ is created, in line 1.
- In the next step, in line 2, the actions are converted to primitive workflow tasks and these tasks are inserted to task set: $T_{w_{f_1}} = \{ \langle \langle T1' \rangle, \{p_{\tau_1}\}, \{o_{\tau_1}\} \rangle, \langle \langle T2' \rangle, \{p_{\tau_2}\}, \{o_{\tau_2}\} \rangle, \langle \langle T3' \rangle, \{p_{\tau_3}\}, \{o_{\tau_3}\} \rangle \}$

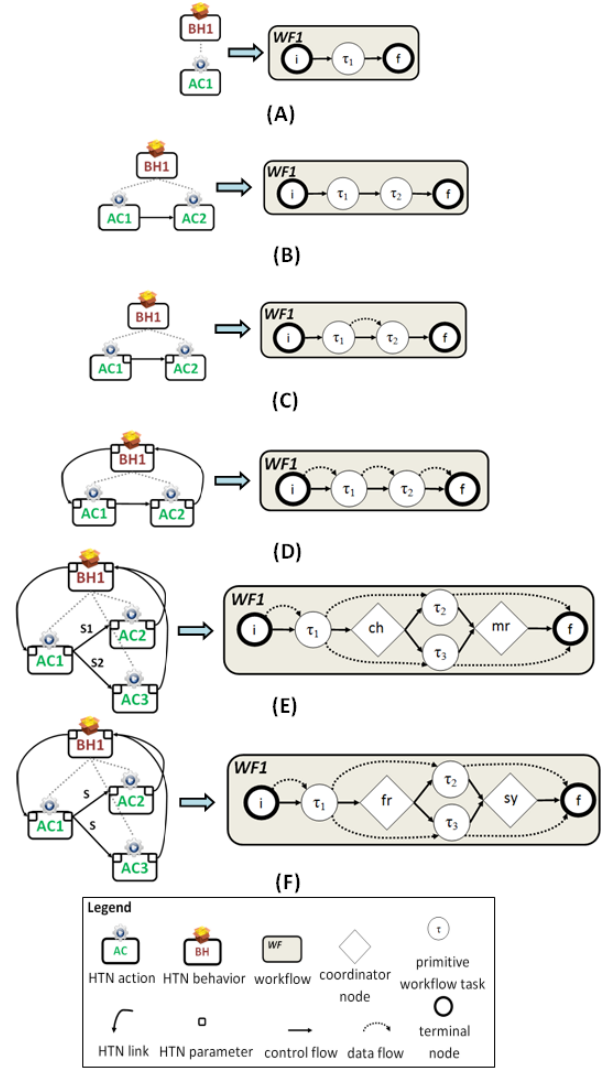


Figure 2. Reduction of Primitive HTN Patterns

- The control flow, $CF_{w_{f_1}} = \{ \langle i_{w_{f_1}}, \tau_1 \rangle, \langle \tau_1, \tau_2 \rangle, \langle \tau_1, \tau_3 \rangle, \langle \tau_2, f_{w_{f_1}} \rangle, \langle \tau_3, f_{w_{f_1}} \rangle \}$, and data flow $DF_{w_{f_1}} = \{ \langle i_{w_{f_1}}, d_{in_{w_{f_1}}} \rangle, \langle \tau_1, d_{in_{\tau_1}} \rangle, \langle \tau_1, d_{out_{\tau_1}} \rangle, \langle \tau_2, d_{in_{\tau_2}} \rangle, \langle \tau_1, d_{out_{\tau_1}} \rangle, \langle \tau_3, d_{in_{\tau_3}} \rangle, \langle \tau_2, d_{out_{\tau_2}} \rangle, \langle f_{w_{f_1}}, d_{out_{w_{f_1}}} \rangle, \langle \tau_3, d_{out_{\tau_3}} \rangle, \langle f_{w_{f_1}}, d_{out_{w_{f_1}}} \rangle \}$ sets are filled in line 3.
- Finally, a fork-synchronizer node pair is inserted to required points, in line 4. This operation fills the coordinator node set $C_{w_{f_2}} = \{ fr_1, sy_1 \}$ and updates the control flow set $CF_{w_{f_1}} = \{ \langle i_{w_{f_1}}, \tau_1 \rangle, \langle \tau_1, fr_1 \rangle, \langle fr_1, \tau_2 \rangle, \langle fr_1, \tau_3 \rangle, \langle \tau_2, mr_1 \rangle, \langle \tau_3, mr_1 \rangle, \langle mr_1, f_{w_{f_1}} \rangle \}$.

B. Correctness of Reduction

Theorem 1: Let β is a behavior defined with HTN semantics. $REDUCE(\beta)$ terminates and returns a workflow wf .

Proof: A behavior represents a tree, and is reduced completely after all subtasks are reduced. So, from the line 2a of algorithm, the algorithm is executed over again for subbehaviors until reaching to the leaf actions. Finally, after

the leaves are transformed in line 2b, algorithm proceeds and bottom-up construction of root behavior is achieved. ■

Theorem 2: Let $B = \{\beta_1, \beta_2, \dots, \beta_n\}$ be a collection of HTN behaviors that conforms our constraints and β be one of these. Let $wfg = REDUCE(\beta)$, then wfg is the workflow which corresponds to behavior β .

Proof: The proof of the theorem is by induction:

Hypothesis For an HTN behavior β , there exists a workflow graph $wf = \langle T_{wf}, C_{wf}, CF_{wf}, DF_{wf}, IC_{wf}, OC_{wf}, TN_{wf} \rangle$ where T_{wf} contains the workflow tasks corresponds to sub tasks of β , CF_{wf} and DF_{wf} contains the flows corresponds to links of HTN, and IC_{wf} and OC_{wf} contains inputs and outputs which corresponds to provisions and outcomes of β .

Base Case Suppose β is a behavior with only one action α_1 as sub task. The reduction of β ends up with a workflow $wf = \langle T_{wf}, \emptyset, CF_{wf}, \emptyset, \emptyset, \emptyset, TN_{wf} \rangle$ where $T_{wf} = \{\tau_1\}$ and $CF_{wf} = \{\langle i_{wf}, \tau_1 \rangle, \langle \tau_1, f_{wf} \rangle\}$. As is seen in line 2b, after workflow is created in line 1, τ_1 is constructed for α_1 and then it is connected with i_{wf} and f_{wf} in line 3. (see Figure-2(A))

Inductive Step Besides the sequence structure in HTN, there exists a few structures such as nesting, conditional branching and parallelism. We will analyze each of the structures case by case to show that results of our translation are correct.

Case 1: While HTN plans can only be extended breadth-wise with primitive subtasks, expansion in depth is provided with subbehaviors. (see Figure-3)

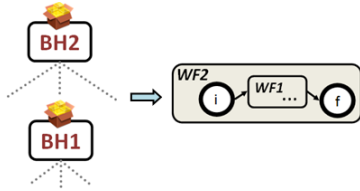


Figure 3. Reduction of nested behavior

Suppose β is a behavior with a subbehavior β_{sub} . From our hypothesis we know that there exists a workflow wf_{sub} for subbehavior β_{sub} . The reduction of β leads to a workflow $wf = \langle T_{wf}, \emptyset, CF_{wf}, \emptyset, \emptyset, \emptyset, TN_{wf} \rangle$ where $T_{wf} = \{wf_{sub}\}$ and $CF_{wf} = \{\langle i_{wf}, wf_{sub} \rangle, \langle \tau_1, wf_{sub} \rangle\}$.

Case 2: Suppose $\beta = \langle BH1', \emptyset, \emptyset, \{\theta_1, \theta_2, \theta_3\}, \Gamma_{\beta_1} \rangle$, where $\Gamma_{\beta_1} = \{\langle \theta_1, \theta_2, S'_1 \rangle, \langle \theta_1, \theta_3, S'_2 \rangle\}$, is a behavior with a conditional branching structure. (similar to Figure-2(E))

From our hypothesis we assume that we have valid workflow tasks τ_1, τ_2, τ_3 which correspond to the HTN tasks $\theta_1, \theta_2, \theta_3$. The behavior β is reduced to a workflow $wf = \langle T_{wf}, C_{wf}, CF_{wf}, \emptyset, \emptyset, \emptyset, TN_{wf} \rangle$ where $T_{wf} = \{\tau_1, \tau_2, \tau_3\}$, $C_{wf} = \{ch_1, mr_1\}$ and $CF_{wf} = \{\langle i_{wf}, \tau_1 \rangle, \langle \tau_1, ch_1 \rangle, \langle ch_1, \tau_2 \rangle, \langle ch_1, \tau_3 \rangle, \langle \tau_2, mr_1 \rangle, \langle \tau_3, mr_1 \rangle, \langle mr_1, f_{wf} \rangle\}$. After raw graph is obtained, choice and merge nodes are inserted to the beginning and the end of the exclusive choice structure in line 4.

Case 3: Suppose $\beta = \langle BH1', \emptyset, \emptyset, \{\theta_1, \theta_2, \theta_3\}, \Gamma_{\beta_1} \rangle$, where $\Gamma_{\beta_1} = \{\langle \theta_1, \theta_2, S'_1 \rangle, \langle \theta_1, \theta_3, S'_1 \rangle\}$, is a behavior with a parallelism structure. (similar to Figure-2(F))

From our hypothesis we know that we have corresponding workflow tasks τ_1, τ_2, τ_3 of the HTN tasks $\theta_1, \theta_2, \theta_3$. The behavior β is reduced to a workflow $wf = \langle T_{wf}, C_{wf}, CF_{wf}, \emptyset, \emptyset, \emptyset, TN_{wf} \rangle$ where $T_{wf} = \{\tau_1, \tau_2, \tau_3\}$, $C_{wf} = \{fr_1, sy_1\}$ and $CF_{wf} = \{\langle i_{wf}, \tau_1 \rangle, \langle \tau_1, fr_1 \rangle, \langle fr_1, \tau_2 \rangle, \langle fr_1, \tau_3 \rangle, \langle \tau_2, sy_1 \rangle, \langle \tau_3, sy_1 \rangle, \langle sy_1, f_{wf} \rangle\}$. In line 4, fork and synchronizer nodes are inserted to the beginning and the end of the parallel split structure in the raw workflow. ■

We proved that our reduction mechanism transforms the HTN plan into the workflow model correctly. In our proof, we showed the correspondence of the result workflow to the input plan.

VI. CASE STUDY

To illustrate the reduction of plans to workflows, a tourism application is implemented as a case study with SEAGENT Framework. In this application an agent which plays tourism agency role is responsible for making a vacation plan. A plan ontology (*BHPlanVacation*) which is the implementation of planning a vacation goal (*PlanVacation*) is provided in the knowledgebase of this agent.

Ontology³ individual of *BHPlanVacation* is depicted in Figure-5(B). *BHPlanVacation* behavior has three subtasks, and it needs location information (*location* provision) and vacation budget (*budget* provision) to make plan. After the execution of the behavior is completed, it gathers the remainder of budget (*remainder* outcome). Firstly, a hotel room is booked in specified holiday resort (*ACBookHotelRoom*), and remainder from budget is passed to the next task. After reservation operation, a travel ticket is bought according to the customer request (*BHBuyTravelTicket*). Finally, a car is rented to go to the hotel from the airport or terminal (*ACRentCar*), and the remainder value is passed to the parent behavior. Representation of the plan which is designed according to our HTN definitions in SEAGENT HTN Editor⁴ is shown in Figure-5(A).

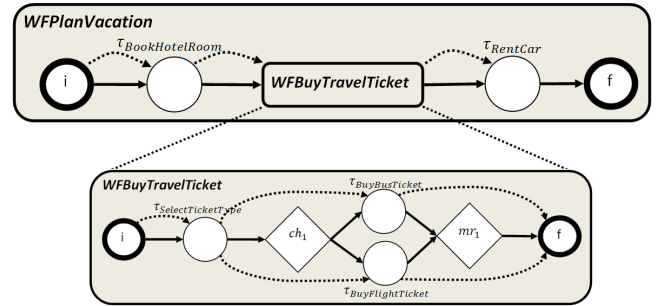


Figure 4. WFPlanVacation workflow

When agent determines to execute the *PlanVacation* goal, it gives the goal description to the planner. After the planner ascertains that the goal is atomic, it searches for the appropriate act in the knowledgebase via the ActMatcher. The ActMatcher finds the *BHPlanVacation* HTN description and

³Full version: <http://www.seagent.ege.edu.tr/etmen/LADS009CaseStudy.zip>

⁴HTN Editor is a part of the SEAGENT Development Environment that is used to build up HTN ontologies easily.

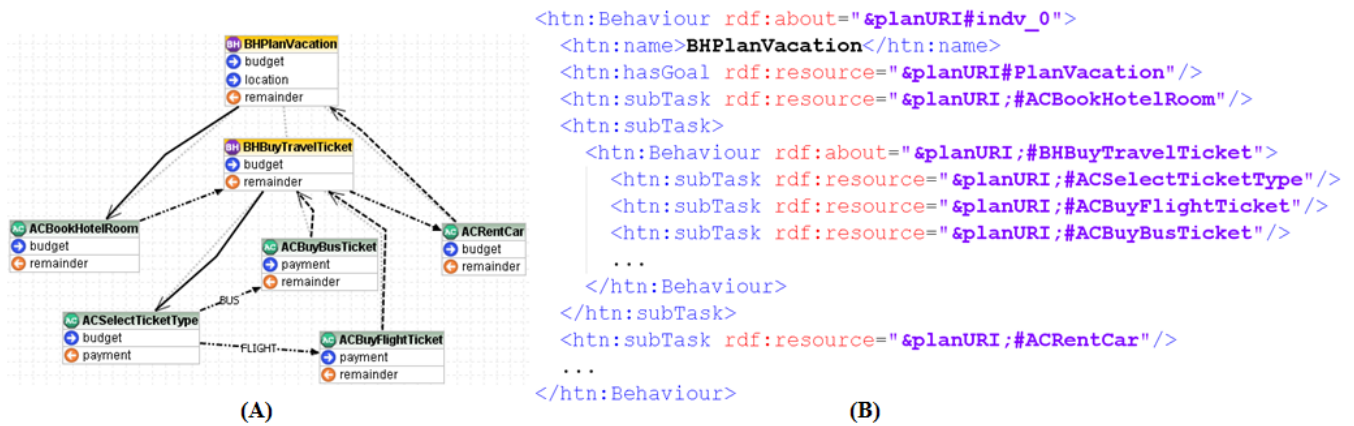


Figure 5. BHPlanVacation A) HTN Representation B) Ontology Individual

transmits it to the HTNReducer to reduce it to workflow. After the HTNReducer completes reduction, the generated workflow is the executable form of the plan description. The workflow which is constructed by HTNReducer is shown in Figure-4.

As is seen in the Figure-4, the workflow tasks are created for all actions of plan and the subbehavior is converted to subworkflow. After the workflow that corresponds to BHPlanVacation is constructed, the planner starts to proceed on the workflow via the ExecutionToken. Tasks are performed when the token visits them. Execution of a task means execution of the java class that is attached to the corresponding HTN action. Java reflection API is used to create and execute action class instances.

VII. CONCLUSION

This paper briefly depicts the architecture of SEAGENT agent development framework's planner. The main characteristic of the proposed architecture is its being based on the workflow technology and its ability to process the artifacts of agent programming paradigms such as plans, services, goals, and roles by executing these artifacts after reducing them to workflows.

To implement the ideas behind the proposed architecture, we described an HTN ontology to define agent plans, developed a workflow component using Java, and focused on the reduction of agent plans to workflows. We used this planner architecture in industrial and academical projects. The last version of the SEAGENT can be downloaded⁵ as an open source project.

SEAGENT planner has been designed with the idea that different plan definition languages other than HTN can also be reduced to the generic workflow model. In addition, business process definition languages such as BPEL, OWL-S can also be reduced to the generic workflow model. Moreover, these business process definition languages can be used in connection with different plan definition languages providing the interoperability of them. These features show the way of incorporating different languages into agent programming paradigms as well as offering a high degree of flexibility in developing agent systems.

⁵SEAGENT Semantic Web Enabled Framework, <http://seagent.ege.edu.tr/>

REFERENCES

- [1] T. Andrews, F. Curbera, H. Dholakia, Y. Golland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovicand, and S. Weerawarana, "Business process execution language for web services v-1.1," W3C, Candidate Recommendation, 2003.
- [2] WfMC, "Workflow management coalition workflow standard: Workflow process definition interface - xml process definition language (xpdl) (wfmc-tc-1025)," Workflow Management Coalition, Lighthouse Point (FL), Tech. Rep., 2002.
- [3] K. Erol, J. Hendler, and D. S. Nau, "Semantics for hierarchical task-network planning," College Park, MD, USA, Tech. Rep., 1994.
- [4] K. Sycara, M. Williamson, and K. Decker, "Unified information and control flow in hierarchical task networks," in *Working Notes of the AAAI-96 workshop 'Theories of Action, Planning, and Control'*, 1996.
- [5] W. Sadiq and M. Orlowska, "Modeling and verification of workflow graphs," in *Technical Report No. 386, Department of Computer Science. The University of Queensland, Australia*, 1996.
- [6] E. Sirin, B. Parsia, D. Wu, J. A. Hendler, and D. S. Nau, "Htn planning for web service composition using shop2," *J. Web Sem.*, vol. 1, no. 4, pp. 377–396, 2004.
- [7] G. Caire, D. Gotta, and M. Banzi, "Wade: a software platform to develop mission critical applications exploiting agents and workflows," in *AAMAS (Industry Track)*, 2008, pp. 29–36.
- [8] A. P. F. Bellifemine and G. Rimassa, "JADE - a FIPA-compliant agent framework," in *Proceedings of the Practical Applications of Intelligent Agents*, 1999.
- [9] E. E. Ekinci, A. M. Tiryaki, Ö. Gürçan, and O. Dikenelli, "A planner infrastructure for semantic web enabled agents," in *OTM Workshops*, 2007, pp. 95–104.
- [10] O. Dikenelli, "Seagent mas platform development environment," in *AAMAS (Demos)*, 2008, pp. 1671–1672.
- [11] J. R. Graham, K. Decker, and M. Mersic, "Decaf - a flexible multi agent system architecture," *Autonomous Agents and Multi-Agent Systems*, vol. 7, no. 1-2, pp. 7–27, 2003.
- [12] K. P. Sycara, M. Paolucci, M. V. Velsen, and J. A. Giampapa, "The retsina mas infrastructure," *Autonomous Agents and Multi-Agent Systems*, vol. 7, no. 1-2, pp. 29–48, 2003.
- [13] W. M. P. van der Aalst, "The application of petri nets to workflow management," *Journal of Circuits, Systems, and Computers*, vol. 8, no. 1, pp. 21–66, 1998.
- [14] S. W. Sadiq, M. E. Orlowska, W. Sadiq, and C. Foulger, "Data flow and validation in workflow modelling," in *ADC*, 2004, pp. 207–214.
- [15] W. Sadiq and M. E. Orlowska, "Analyzing process models using graph reduction techniques," *Inf. Syst.*, vol. 25, no. 2, pp. 117–134, 2000.
- [16] J. Davis, W. Du, and M.-C. Shan, "Openpm: An enterprise process management system," *IEEE Data Eng. Bull.*, vol. 18, no. 1, pp. 27–32, 1995.
- [17] W. Du, J. Davis, and M. C. Shan, "Flexible specification of workflow compensation scopes," in *GROUP '97: Proceedings of the international ACM SIGGROUP conference on Supporting group work*. New York, USA: ACM, 1997, pp. 309–316.
- [18] B. K. W.M.P van der Aalst, A.H.M. ter Hofstede and A. Barros, "Workflow patterns," in *Distributed and Parallel Databases*, July 2003, pp. 5–51.

The ARTS Real-Time Agent Architecture

Konstantin Vikhorev
 School of Computer Science
 University of Nottingham
 Nottingham, NG8 1BB, UK
 Email: kxv@cs.nott.ac.uk

Natasha Alechina
 School of Computer Science
 University of Nottingham
 Nottingham, NG8 1BB, UK
 Email: nza@cs.nott.ac.uk

Brian Logan
 School of Computer Science
 University of Nottingham
 Nottingham, NG8 1BB, UK
 Email: bsl@cs.nott.ac.uk

Abstract—We present a new approach to providing soft real-time guarantees for Belief-Desire-Intention (BDI) agents. We define what it means for BDI agents to operate in real time, or to satisfy real-time guarantees. We then develop a model of real-time performance which takes into account the time by which a task should be performed and the relative priority of tasks, and identify the key stages in a BDI architecture which must be bounded for real-time performance. As an illustration of our approach we introduce a new BDI architecture, ARTS, which allows the development of agents that guarantee (soft) real-time performance. ARTS extends ideas from PRS and JAM to include goals and plans which have deadlines and priorities, and schedules intentions so as to achieve the largest number of high priority intentions by their deadlines.

Index Terms—BDI Agents, Real-time guarantees, Task Scheduling, Priority, Deadline, ARTS.

I. INTRODUCTION

The design of an agent system which can operate effectively in a real-time dynamic environment is a major challenge for multiagent research. The main difficulty in building real-time agent systems is how to specify real-time constraints and how to ensure that the agent system meets these constraints. As with other computational systems, agents are *resource bounded* because their processors have limited speed and memory. Traditionally, agents have been developed without much attention to resource limitations. However such limitations become important when an agent system operates in a dynamic environment. The reasoning processes implicit in many agent architectures may require significant time to execute (in some cases exponential time), with the result that the environment may change while the agent makes a decision about which activity to pursue. Thus a decision made by the agent may be wrong (incorrect, sub-optimal, or simply irrelevant) if it is not made in a timely manner.

A number of agent architectures and platforms have been proposed for the development of agent systems which must operate in highly dynamic environments. For example, the Procedural Reasoning System (PRS) [1] and PRS-like systems, e.g., PRS-CL [2], JAM [3], SPARK [4] have features such as metalevel reasoning which facilitate the development of agents for real time environments. However, to provide real time guarantees, these systems have to be programmed for each particular task environment—there are no general methods or tools which allow the agent developer to specify that a particular goal should be achieved by a specified time or that

an action should be performed within a particular interval of an event occurring. Rather each application has to be individually tuned by the developer. There are also a number of hybrid agent architectures such as ROACS [5] and SIMBA [6]. A hybrid architecture consists of an AI subsystem and a low-level control subsystem connected by communication interface. Such systems attempt to improve responsiveness by separating the ‘real-time’ aspects of the architecture from the high-level control. However while such systems can simplify the development of agents for real-time environments, they provide limited high-level support for managing the timely execution of tasks.

In this paper we present a new approach to Belief-Desire-Intention (BDI) architectures for real-time agents. We develop a model of real time performance which takes into account the time by which a task should be performed and the relative priority of tasks, and identify the key stages in a BDI architecture which must be bounded for real time performance. As an illustration of our approach we introduce a new BDI architecture, ARTS, which allows the development of agents that guarantee (soft) real time performance. ARTS extends ideas from PRS and JAM to include goals and plans which have *deadlines* and *priorities*, and schedules intentions so as to achieve the largest number of high priority intentions by their deadlines.

The remainder of the paper is organised as follows. In section 2 we develop a notion of ‘real-time’ appropriate to agent-based systems. In section 3 we present our method for any BDI architecture. In section 4 to illustrate our approach the new real-time agent architecture ARTS is introduced. In section 5 we compare our methods with related approaches. And, finally, in section 6 we conclude and outlined directions for future research. discussion.

II. REAL-TIME GUARANTEES

In real-time programming a distinction is made between *hard real-time* and *soft real-time* systems. In the context of agent systems, hard real-time means that the agent must process its inputs (i.e., facts and goals) and produce a response within a specified time. For an agent system which provides hard real-time guarantees there is therefore a strict upper bound on the time to process incoming information and produce a response. In soft real-time, the agent may not produce a response within the specified time in all cases, i.e. timeliness

constraints may be violated under load and fault conditions without critical consequences.¹ For BDI agents, we would argue that the relevant notion of ‘response’ is the achievement of a high level goal. However, for agents in open environments, providing hard real-time guarantees for anything other than the internal operations of the agent is typically not possible, unless we make strong assumptions about the characteristics of the agent’s environment. In this paper we therefore focus on soft real-time guarantees for achieving the agent’s top level goals.

We assume that each of the agent’s top level achievement goals is associated with a (possibly infinite) *deadline* which specifies the time by which the goal should be achieved. A set of goals which can all be achieved by the deadlines is termed *feasible*. Which sets of goals are feasible will depend on the speed at which the environment changes, the capabilities of the agent etc. In general, it may not be possible to achieve all of an agent’s goals by their deadlines. For example, goals produced by users or other agents, or autonomously generated in response to an event in the agent’s environment, may result in a previously feasible set of goals becoming infeasible, if there is insufficient time to achieve each goal, or an agent may have no plan to achieve a particular goal. In such situations, it is frequently more important to achieve some goals than others. For example, the goal of submitting a conference paper on time may be more important than a goal to get coffee before the coffee shop closes. We therefore assume that each goal is associated with a *priority* which specifies the importance of achieving the goal. Priorities define a total preorder, \preceq , over goals. A set of goals g is said to be maximal if it is feasible and there is no other set of goals g' such that $g' \preceq g$ for some suitable lifting of \preceq to sets of goals. We define a *real-time BDI agent* as an agent which achieves a maximal set of goals, i.e., the largest number of high priority goals by their deadlines.

III. CHANGES TO THE BDI ARCHITECTURE

In this section we outline the changes necessary to a BDI architecture to implement a *real-time BDI agent*. We assume a simple generic BDI architecture in which an agent has beliefs and goals, and selects plans (sequences of subgoals and primitive actions) in order to achieve its goals or in response to new beliefs. Once the agent has adopted a plan it becomes an intention, and at each cycle the agent executes a single step of one of its current intentions. To implement real-time BDI agents within such an architecture, two main changes are required: we must add additional information about goals and plans to support real time guarantees, and we need to change the BDI execution cycle to ensure that the agent’s cycle time is bounded and that the maximum number of high priority

goals are achieved by their deadlines. We consider each of these changes in turn below.

A. Additional Information

As discussed above, in order to provide real-time guarantees, each top-level goal must be associated with a *deadline* which specifies the time by which the goal should be achieved. We assume that the deadline for a goal is specified when the goal is generated by a user (or another agent), and is expressed as a real time value in some appropriate units (milliseconds, minutes, hours etc.). By default, the plan selected to achieve a top-level goal (and its subgoals and subplans) inherit the deadline of the top-level goal. However we allow the deadline of the intention associated with the top-level goal to be advanced by a plan, if the execution context of the plan is such as to suggest that an earlier deadline should be adopted for the goal. For example, a subgoal in a plan may be to buy an item; if the store which sells the item is about to close, the deadline of the of the intention may be advanced for the duration of the ‘buy an item’ subgoal so that it is achieved while the store is open. For intentions associated with *fact-invoked* plans (i.e., plans triggered by the agent’s beliefs), the default deadline is infinity, as no *a priori* deadline can be associated with a fact independent of a belief context. However, as with top-level goals, we allow the deadline of intentions associated with fact-invoked plans to be advanced by the invoked plan, if if the execution context of the plan is such as to suggest that the response to the new fact warrants an earlier deadline.²

Each top-level goal is also associated with a *priority* (e.g., a non-negative integer value, with larger values taken to indicate higher priority) which specifies the relative importance of achieving the goal. Each plan also has a *plan priority* which specifies the relative utility of the plan for the triggering goal or belief. We assume that the agent always intends plans with the highest priority and that goal and plan priorities are commensurable (i.e., that the order over intention priorities is independent of the belief context). For intentions associated with *fact-invoked* plans, the priority is the plan priority of the invoked plan.

Each plan is also associated with a *duration*, an estimate of the real time necessary to execute the plan. In order to define durations, we assume that each primitive action has a *timeout* (specified by the agent developer) which defines the maximum amount of real time required to perform the action. Actions which do not complete by their timeout are assumed to have failed. To the duration of an action we add the upper bound on the deliberation cycle (see section III-B below) to give the maximum amount of time necessary to select and execute the action. The duration of a non-primitive activity within a plan is the sum of the durations of its subplans (i.e., the duration of a top-level plan is the sum of the durations of all subplans intended for the goal). Assuming the plan library is of fixed size, we compute the durations of subplans as follows.

¹Some computer systems (for example, real-time video) utilise a stricter notion of real-time guarantees, where the precise time at which a response is produced matters [7], [8]. Hard real-time for this type of system requires a response at an exact time rather than before a deadline, and soft real-time means that the response time lies within a defined uncertainty range around the required time. However we would argue that, in agent based systems, this stricter sense of real time guarantee is less appropriate; for many task environments, the key requirement is that a goal is achieved *before* a deadline rather than exactly at a specific time.

²For how deadlines are (re)computed in the ARTS architecture, see section IV-C.

- 1) For every agent's plan, we compute all possible variants of an intention, leading by this plan. This can be represented as a tree structure. For the moment, we assume that there are no loops or recursion within plans.
- 2) Leaf plans do not contain calls to other plans and include only the addition and deletion of goals and primitive actions, and their duration can be easily calculated from the time required for basic agent actions (see below) and the timeouts on primitive actions.
- 3) Starting from leaf plans we can estimate the duration of each intention variant. The maximum and the minimum duration are the upper and the lower bound of the plan duration.

In case of plans with loops with undefined number of repetitions or recursion within the plan, the minimum duration is the shortest path through the tree structure and the maximum duration is infinity. In most cases, especially in a complex system, we will not be able to provide the exact upper bound estimation of duration.

B. Changes to the BDI Execution Cycle

We assume that the internal operations of the agent—adding or deleting a belief or goal, selecting a plan, adopting an intention, selecting an intention to execute and executing a single step of the intention—require time bounded by the size of the agent's program and its beliefs and goals. Adding or deleting a belief or goal, adopting an intention, and executing a single step of an intention can be assumed to take constant time. However selecting a plan and intention to execute are intractable in the general case, and it is necessary to approximate the choices of an unbounded agent to limit the agent's cycle time.

To bound the time necessary to select a plan, we assume that goals and plans are processed in order of priority. That is, for each goal in priority order, the highest priority plan for that goal is checked to see if it is both executable in the current belief context and feasible (has a duration less than the deadline of the triggering goal). If the plan is executable and feasible, the agent immediately commits to the plan and processing moves to the next goal. If the plan is not executable or feasible matching continues for the current goal with the next plan in priority order. Plan selection stops when all goals have an executable feasible plan or a user definable *plan selection timeout* is reached. At this point the agent has zero or more executable, feasible plans, which are merged into the intention structure, either as new top-level intentions (for plans triggered by new top-level goals or facts), or by adding them to existing intentions.

To bound the time necessary to select an intention to execute at the current cycle, we utilise a deadline monotonic scheduling algorithm which, while not optimal, gives preference to urgent, high-priority intentions:

- 1) find the highest priority feasible intention, i.e., where the remaining execution time is less than the deadline;
- 2) find the next most important intention which is feasible for the existing schedule and assuming that tasks are

- executed in deadline order, earliest deadline first;
- 3) repeat 2 until no more intentions can be scheduled;
- 4) execute the next step of the first intention in the schedule.

An intention is feasible if it can be inserted in the schedule in deadline order while meeting its own and all currently scheduled deadlines. If all intentions in the schedule had the same priority, then the resulting schedule must be feasible if any schedule is, i.e., if a system is unschedulable with deadline monotonic ordering then it is unschedulable with all other orderings [9]. This algorithm has a worst case complexity of $O(n)$, where n is the number of the agent's intentions.

There are two possible scheduling scenarios: pessimistic, which is based on the maximum duration of a plan and optimistic, which is based on the minimum duration. Pessimistic scheduling has limited applicability, as in most cases the maximum duration is equal to infinity. In many cases, it is therefore more reasonable to implement an optimistic scheduler as this places no restrictions on the structure of plans. With optimistic scheduling, even if the maximum duration of a plan is infinite, it may still be scheduled, but can be descheduled if it becomes infeasible (i.e., if the minimum duration of the plan is greater than the deadline of the triggering goal given the currently scheduled plans).

IV. ARTS: AGENT REAL-TIME SYSTEM

In this section ARTS, an implementation of the real-time BDI agent architecture described above. ARTS is an agent programming framework for agents with soft real-time guarantees; an ARTS agent will attempt to achieve as many high priority tasks by their specified deadlines as possible. The syntax and execution semantics of ARTS is based that of PRS-CL [2] and JAM [3], augmented with information about deadlines, priorities, and durations, and changes to the interpreter to implement time bounded priority driven plan selection and deadline monotonic intention scheduling. ARTS is implemented in Java, and the current prototype implementation includes the core language described below, and implementations of some basic primitive actions. Additional user-defined primitive actions can be added using a Java API. In the interests of brevity, we have omitted the meta-level features of ARTS.

An ARTS agent consists of five main components: a database, a goal stack, a plan library, an intention structure, and an interpreter. The database contains the agent's current beliefs (facts). The goal stack is a set of goals to be realised. The plan library contains a set of plans which can be used to achieve agent's goals or react to particular situations. The intention structure contains plans that have been chosen to achieve goals or respond to facts. The interpreter is the main component of the agent. It manipulates the agent's database, goal stack, plan library and intention structure and reasons about which plan to select based on the agent's beliefs and goals to create and execute intentions. Changes to the agent's environment or posting of new goals invokes reasoning to search for plans that might be applied to the current situation. The ARTS interpreter

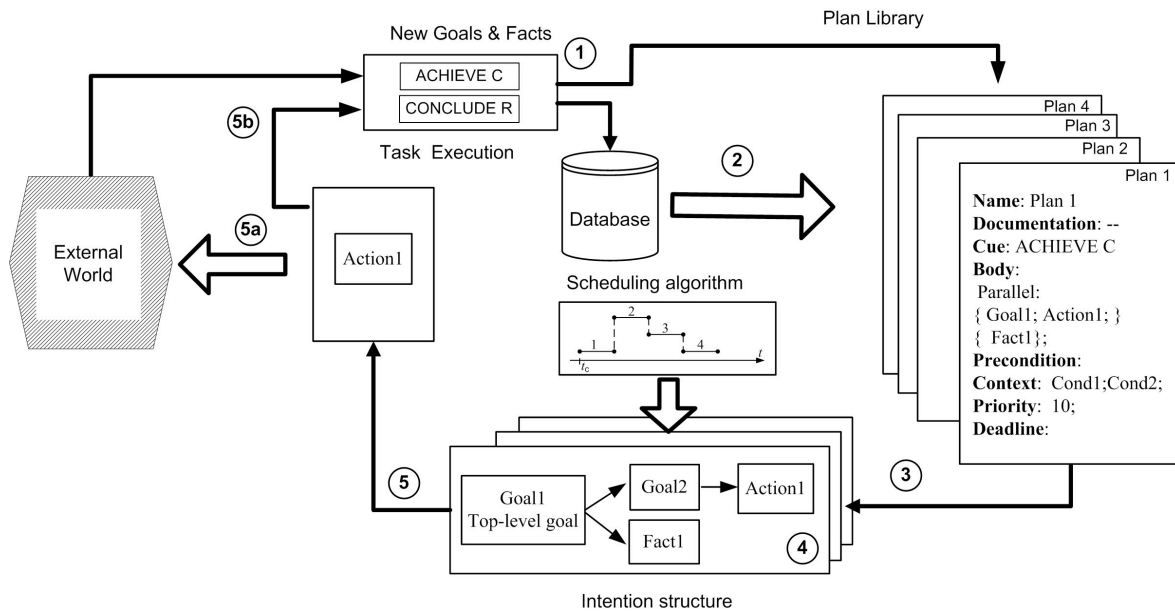


Fig. 1. The execution cycle of ARTS agent

selects one plan from the list of applicable plans, intends and schedules it, and executes the next step of first intention in the computed schedule.

A. Facts

The database of an ARTS agent contains facts (beliefs) that represent the state of the agent and its environment. Facts may represent information about state variables, sensory input, derived information or information about other agents, etc.

```

fact      ::= wff
wff       ::= pred_name term_exp* | (NOT wff)
           | (AND wff+) | (OR wff+)
term_exp  ::= value | variable | function
value     ::= integer | float | string
variable  ::= "$"var_name
function  ::= (fun_name term_exp+)
    
```

where *pred_name*, *fun_name* and *var_name* name predicated, functions and variables respectively.

B. Goals

ARTS distinguishes two categories of goals: top-level goals and subgoals. ARTS supports two top-level goal operators: **ACHIEVE** and **CONCLUDE**. An **ACHIEVE** goal specifies that the agent desires to achieve a particular goal state. A **CONCLUDE** goal inserts a certain fact into the database and possibly invokes a fact-invoked plan. The form of top-level

goals is given by:

```

goal_exp ::= achieve | conclude
achieve  ::= "ACHIEVE" wff [pr] [dl]
           { [by] | [not_by] } ";"
conclude ::= "CONCLUDE" wff { [by] | [not_by] } ";"
pr        ::= ":PRIORITY" ground_term
dl        ::= ":DEADLINE" ground_term
by        ::= ":BY" plan_name+
not_by    ::= ":NOT_BY" plan_name+
    
```

where *plan_name* is the name of a plan. The **:PRIORITY** field of an **:ACHIEVE** top-level goal is optional and allows the specification of either a constant priority or an expression which allows the calculation of the plan priority as function of plan variables (see below). The default priority of a top-level goal is zero. The **:DEADLINE** field is also optional and allows the specification of the deadline of the goal. By default the deadline is equal to infinity. **CONCLUDE** goals have zero priority and an infinite deadline.

The developer can specify one or more top-level goals for the agent as part of the agent's program using the keyword "GOALS:". For example:

```

GOALS:
  ACHIEVE PrepareLecture agents101 : PRIORITY 9 :DEADLINE 50;
  ACHIEVE HaveLunch :PRIORITY 7 :DEADLINE 40;
  ACHIEVE BorrowBook R&N :PRIORITY 2 :DEADLINE 30;
    
```

Subgoals are goals generated within plans. ARTS has the following subgoals operators:

```

ACHIEVE C    achieve condition C
CONCLUDE F   add fact F to the database
TEST C       test for the condition C
RETRACT F    retract fact F from database
WAIT C       wait until condition C is true
    
```

In contrast to top-level goals, the deadline and priority of

ACHIEVE subgoals are not specified but inherited from the plan containing the subgoal.

C. Plans

Plans define a procedural specification for achieving a goal. In specifying plans we distinguish between *plan trigger variables* and *plan body variables*. Plan trigger variables are free variables appearing in the cue, precondition and context fields, while plan body variables are variables appearing in the body of the plan. Plan trigger variables must be ground when the plan is selected, while binding of plan body variables can be deferred to the execution of the corresponding plan step. The agent's plan library is introduced by the keyword "PLANS:" followed by a list of plans of the form:

Name is an unique symbolic identifier of the Plan.

Documentation is an optional field which used to store a descriptive text string.

Cue specifies the purpose of the Plan and is used to select the plan for possible execution. The **Cue** field can contain either an ACHIEVE or CONCLUDE goal. A ACHIEVE goal in the **Cue** field means that the Plan may be used to achieve some condition, while a CONCLUDE goal means that the Plan may be chosen for possible execution when a fact is added to the database.

Precondition specifies conditions that must be satisfied for Plan to be applicable. This field is optional and can contain both ACHIEVE and TEST goal expressions. An ACHIEVE G precondition means that the system must currently have G as a goal in order for the Plan to be applicable, while a TEST C precondition means that C must be true for the Plan to be applicable.

Context defines additional conditions (i.e. ACHIEVE and TEST goal expressions) on plan execution. This field is optional and has similar functionality to the Precondition field, but in contrast to the precondition it must be satisfied before and during Plan execution. As in JAM, this significantly increases the reactivity of the agent.

Body defines a sequence of simple activities (i.e. primitive actions, addition and deletion of goals and facts), and complex constructs (e.g. loops, (non)deterministic choice, etc, see below).

Priority specifies the relative utility of the Plan. The plan priority is used to choose between the applicable plans for a particular goal. The priority field is optional and allows the specification of either a constant priority or an expression which allows the calculation of the plan priority as function of variables appearing in the plan trigger. The default priority value is 0.

Deadline specifies a deadline for the plan. The deadline field is optional and allows programmer to advance the deadline inherited from the triggering goal. The deadline can be specified as a constant value or an expression which allows the calculation of the plan deadline as function of variables appearing in the plan trigger. If the specified plan deadline is earlier than the deadline for this intention it becomes the deadline for the intention during the execution of the plan (i.e.,

<i>plan</i>	::=	"PLAN: { <i>p_name</i> [<i>p_doc</i>] <i>p_cue</i> [<i>p_precond</i>] [<i>p_cont</i>] <i>p_body</i> [<i>p_pr</i>] [<i>p_dl</i>] [<i>p_attr</i>] ""}"
<i>p_name</i>	::=	"NAME:" string";"
<i>p_doc</i>	::=	"DOCUMENTATION:" [string]";"
<i>p_cue</i>	::=	"CUE:" <i>p_goal_exp</i> "";"
<i>p_precond</i>	::=	"PRECONDITION:" <i>p_cond</i> * "";"
<i>p_cont</i>	::=	"CONTEXT:" <i>p_cond</i> * "";"
<i>p_body</i>	::=	"BODY:" <i>body_elem</i> *
<i>p_pr</i>	::=	"PRIORITY:" <i>term_exp</i> "";"
<i>p_dl</i>	::=	"DEADLINE:" <i>term_exp</i> "";"
<i>body_seq</i>	::=	"{" <i>body_elem</i> * "}"
<i>body_elem</i>	::=	<i>activity</i> <i>b_and</i> <i>b_or</i> <i>b_parallel</i> <i>b_do_all</i> <i>b_do_any</i> <i>b_do_while</i> <i>b_while</i> <i>b_when</i>
<i>activity</i>	::=	<i>prim_act</i> <i>misc_act</i> <i>subgoal</i> "";"
<i>b_and</i>	::=	"AND:" <i>body_seq</i> +"";"
<i>b_or</i>	::=	"OR:" <i>body_seq</i> +"";"
<i>b_parallel</i>	::=	"PARALLEL:" <i>body_seq</i> +"";"
<i>b_do_all</i>	::=	"DO_ALL:" <i>body_seq</i> +"";"
<i>b_do_any</i>	::=	"DO_ANY:" <i>body_seq</i> +"";"
<i>b_do_while</i>	::=	"DO:" <i>body_seq</i> "WHILE:" <i>p_cond</i> "";"
<i>b_while</i>	::=	"WHILE:" <i>p_cond</i> <i>body_seq</i> "";"
<i>b_when</i>	::=	"WHEN:" <i>p_cond</i> <i>body_seq</i> "";"
<i>p_goal_exp</i>	::=	"ACHIEVE" <i>wff</i> "CONCLUDE" <i>wff</i>
<i>p_cond</i>	::=	"ACHIEVE" <i>wff</i> "TEST" <i>wff</i>
<i>subgoal</i>	::=	<i>subgoal_op</i> <i>wff</i> "";"
<i>subgoal_op</i>	::=	"ACHIEVE" "CONCLUDE" "TEST" "RETRACT" "WAIT"
<i>prim_act</i>	::=	"EXECUTE:" <i>function</i> [{"TIMEOUT" <i>ground_term</i>]
<i>misc_act</i>	::=	"ASSIGN:" <i>term_exp</i> <i>term_exp</i>

TABLE I
PLAN BNF

it effectively advances the deadline for this intention during the execution of the plan). If the specified deadline is later than the deadline for the intention, the plan deadline is ignored.

ARTS, like JAM, supports standard programming constructs such as DO ...WHILE (loop with postcondition) and WHILE construct (loop with precondition), choice constructs specified by OR (do any in order), AND (do all in order), DO_ALL (do all randomly), DO_ANY (do any randomly), WHEN (conditional execution), and ASSIGN (assignment to plan body variables). The BNF for plans is given in table I.

D. Primitive Actions

The subgoal operators are implemented directly by the ARTS interpreter. Other primitive actions are implemented as Java methods. Each primitive action referenced in a plan body must have Java code which implements the necessary functionality. ARTS supports two mechanisms for defining primitive actions: writing a class which implements the `PrimitiveAction` interface, and direct invocation of methods in existing legacy Java code. Primitive actions are executed by using an EXECUTE action.

In contrast to PRS-CL and JAM, ARTS allows the agent programmer to specify a timeout for each primitive action by using the TIMEOUT keyword. The timeout specifies the maximum amount of real time required to perform the action. Actions which do not complete by their timeout are assumed to have failed. For example:

```
EXECUTE move_to $x $y :TIMEOUT 50
```

E. Interpreter

The ARTS interpreter repeatedly executes the set of activities shown in Figure 1.

- 1) New goals are added to the goal stack and facts corresponding to CONCLUDE goals and external events are added to the database.
- 2) The precondition and context expressions of plans with a cue matching a goal on the goal stack are evaluated against the database to determine if the plan is applicable in the current situation. Goals and plans are matched in priority order as described in section III-B. For ACHIEVE goals, the interpreter checks to see whether the goal has already been accomplished before trying to invoke a plan.
- 3) The resulting set of applicable plans are placed on the intention structure.
- 4) Intentions are scheduled according to their deadline and priority value as described in section III-B. Intentions which are not schedulable, i.e., their minimum remaining execution time is greater than the time remaining to their deadline, are either dropped or have their priority reduced to zero.³
- 5) Finally, the interpreter selects the first intention from the computed schedule and executes the one step of that intention. The result of the execution can be (5a) execution of a primitive action or (5b) the posting of a new subgoal or the conclusion of some new fact.

F. Example

In this section sketch as simple example ARTS agent and show how it allows the specification of soft real-time requirements. The agent has three goals: preparing a lecture, having lunch and picking up a book from the library. Each task has a different priority and deadline. For simplicity, we assume that actions never fail and that the unit of time is the minute.

The algorithm for estimating the duration of a plan is executed when the agent is initialised. Plans have following maximum and minimum durations: $d_1 = 35min$, $d_2 = 12min$, $d_3 = 2min$, $d_4 = 20min$. Once the durations of plans have been estimated, the agent begins the reasoning cycle. The interpreter parses the initial top-level goals. It then attempts to match them against the plan library in order to invoke suitable plans. As a result **Plan 1**, **Plan 2** and **Plan 4** are added to the intention structure. As explained above, plans inherit their deadline and priority values from the triggering goal. This means that the intention related to the prepare lecture goal has the highest priority (9), the intention which corresponds to the goal to have lunch has medium priority (7), and the last intention related to the goal to pickup a book from the library has the lowest priority (2). The scheduling algorithm checks feasibility of each intention before adding them to the schedule. The first most important intention is inserted into schedule, because it is currently empty. Then the feasibility

³This choice is currently determined by a global flag, rather than per goal.

```

GOALS:
  ACHIEVE PrepareLecture agents101 :PRIORITY 9 :DEADLINE 50;
  ACHIEVE HaveLunch :PRIORITY 7 :DEADLINE 40;
  ACHIEVE BorrowBook R&N :PRIORITY 2 :DEADLINE 30;

  CONCLUDE LectureNotes agents101 myNotes;

PLAN: {
  NAME: "Plan 1";
  DOCUMENTATION: "Prepare for lecture";
  CUE: ACHIEVE PrepareLecture $x;
  PRECONDITION: TEST LectureNotes $x, $y;
  BODY:
    EXECUTE revise-lecture $y :TIMEOUT 35;
}

PLAN: {
  NAME: "Plan 2";
  DOCUMENTATION: "Pickup a book from the library";
  CUE: ACHIEVE BorrowBook $x;
  BODY:
    EXECUTE goto library :TIMEOUT 10;
    ACHIEVE Pickup $x;
}

PLAN: {
  NAME: "Plan 3";
  DOCUMENTATION: "Pick up something";
  CUE: ACHIEVE Pickup $x;
  BODY:
    EXECUTE pickup $x :TIMEOUT 2;
}

PLAN: {
  NAME: "Plan 4";
  DOCUMENTATION: "Have lunch";
  CUE: ACHIEVE HaveLunch;
  BODY:
    EXECUTE eat-sandwich :TIMEOUT 20;
}

```

EXAMPLE ARTS AGENT

of the *HaveLunch* intention is checked. It is obvious that the intention is infeasible, because it can't be inserted to the schedule in deadline order together with first intention and it will be dropped. On the other hand the low priority intention is feasible and can be scheduled in deadline order together with the first one. It is important to note that the low priority task will be executed first, because it has an earlier deadline. Once the schedule has been computed, the interpreter executes one step of the first task, i.e., *goto* primitive action, and starts a new cycle.

At the second cycle the interpreter executes next step, i.e. the ACHIEVE *Pickup* goal. The goal invokes **Plan 3** which inherits the deadline of 30 and priority of 2 from the top-level goal and extends the existing intention. The interpreter then performs one step of the plan 3. In the same way it performs action to revise the notes for the lecture from the next intention.

V. RELATED WORK

The scheduling of intentions is key to realisation of real-time agents and a variety of intention scheduling approaches have been explored in the literature. For example, AgentSpeak(XL) [10] and the Soft Real-Time Agent Architecture

[11] use the TÆMS (Task Analysis, Environment Modelling, and Simulation) domain-independent framework [12] together with Design-To-Criteria scheduling [13]. The TÆMS framework assumes a worth-oriented environment, in which goals are associated with a degree of achievement (i.e., a goal may be not fully achievable or not achievable at all). The TÆMS framework allows modelling tasks with deadlines. The problem of using DTC for scheduling agent intentions is that an agent which implements DTC will not perform tasks in some fixed order and is unable to compute a set of feasible tasks because the decision about which task (intention) will be executed is based on some rating (real value between 0 and 1), which changes from cycle to cycle. Astefanoaei *et al.* [14] extend the programming language BUPL to allow agents to synchronise their actions by providing each agent with a set of clock variables. The clock value is used as a condition for executing an action (or waiting). It is not related to the agent's deliberation cycle nor is it used for scheduling of a set of possible actions.

There has been considerable work on approaches to the development of reasoning systems which must operate in highly dynamic environments, e.g., [15], [16], [2], [3], [4]. Much of this work has focused on deliberation and reasoning strategies involving metalevel plans and facts for reasoning about applicable plans, the failure to achieve a goal, changing the intention structure according to user specified criteria, etc. While metalevel reasoning provides great flexibility to the agent developer, it can be complex and has to be programmed for each particular application. In contrast, ARTS has its own well defined real-time reasoning mechanism for tasks with different priorities and deadlines, which does not require utilisation of metalevel capabilities.

VI. CONCLUSION

The main contributions of this paper are an analysis of the meaning of real-time guarantees for a BDI agent, and a proposal for a new BDI agent architecture, ARTS, for the development of real-time BDI agents. ARTS is influenced by the PRS family architectures, such as PRS-CL and JAM. However, unlike previous PRS-like architectures, ARTS includes a duration estimation algorithm, priority driven plan selection and a deadline monotonic intention scheduling algorithm. These features enable an ARTS agent to produce an intention schedule which achieves the greatest number of high priority goals by their deadlines. While the resulting schedule is not necessarily optimal, it is computable in bounded time, and we believe that the kind of "optimistic bounded rationality" implemented by the ARTS architecture provides a simple, predictable framework for agent developers, facilitating the development of agents which can perform tasks of different complexity and scale while providing timely responses to events in highly dynamic environments.

The current ARTS implementation has a number of limitations. For example, the architecture currently assumes that the agent must wait for the completion of each plan step before recomputing the intention structure, i.e., the agent

can't execute intentions in parallel. For plans containing asynchronous primitive actions or WAIT goals, this is clearly not the case. In future work, we plan to extend the scheduler to handle asynchronous execution of intentions. Other directions for future work include improved algorithms for duration estimation and improvements to the basic BDI interpreter to reduce the overall cycle time. It would also be interesting to explore probabilistic scheduling based on the most likely execution time of a plan as opposed to simply the lower and upper bound.

REFERENCES

- [1] M. P. Georgeff and A. L. Lansky, "Procedural knowledge," in *IEEE*, vol. 74, no. 10. IEEE Press, 1987, pp. 1383–1398.
- [2] K. L. Myers, *PRS-CL: A Procedural Reasoning System. User's Guide.*, SRI International, Center, Menlo Park, CA, March 2001.
- [3] M. J. Huber, "JAM: A BDI-theoretic mobile agent architecture," in *Proceedings of The Third International Conference on Autonomous Agents*, Seattle, WA, 1999, pp. 236–243.
- [4] D. Morley and K. Myers, "The spark agent framework," in *Proc. of the Third Int. Joint Conf. on Autonomous Agents and Multi Agent Systems (AAMAS-04)*, New York, NY, July 2004, pp. 712–719.
- [5] J. S. Gu and C. W. de Silva, "Development and implementation of a real-time open-architecture control system for industrial robot systems," *Engineering Applications of Artificial Intelligence*, vol. 17, no. 5, pp. 469 – 483, 2004.
- [6] C. Carrascosa, J. Bajo, V. Julian, J. M. Corchado, and V. Botti, "Hybrid multi-agent architecture as a real-time problem-solving model," *Expert Systems Applications*, vol. 34, no. 1, pp. 2–17, 2008.
- [7] J. Chakareski, J. Apostolopoulos, and B. Girod, "Low-complexity rate-distortion optimized video streaming," in *Proceedings of the International Conference on Image Processing (ICIP)*, vol. 3, Oct. 2004, pp. 2055–2058.
- [8] S. G. Deshpande, "High quality video streaming using content-aware adaptive frame scheduling with explicit deadline adjustment," in *MM '08: Proceeding of the 16th ACM international conference on Multimedia*. New York, NY, USA: ACM, 2008, pp. 777–780.
- [9] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *JACM*, vol. 20, no. 1, pp. 46–61, 1973.
- [10] R. Bordini, A. Bazzan, R. de, O. Jannone, D. Basso, R. Vicari, and V. Lesser, "Agentspeak(XL): efficient intention selection in BDI agents via decision-theoretic task scheduling," in *In Proc. of AAMAS'02*, 2002, pp. 1294–1302.
- [11] R. Vincent, B. Horling, V. Lesser, and T. Wagner, "Implementing soft real-time agent control," in *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*. New York, NY, USA: ACM, 2001, pp. 355–362.
- [12] K. S. Decker and V. R. Lesser, "Quantitative modeling of complex environments," *International Journal of Intelligent Systems in Accounting, Finance and Management*, vol. 2, p. 215234, 1993.
- [13] T. Wagner, A. Garvey, and V. Lesser, "Criteria-directed heuristic task scheduling," *International Journal of Approximate Reasoning*, vol. 19, pp. 91–118, July 1998.
- [14] L. Astefanoaei, F. S. de Boer, and M. Dastani, "On coordination, autonomy and time," in *Proceedings of 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, vol. 2, Budapest, Hungary, 2009, pp. 1357–1358.
- [15] M. P. Georgeff and A. L. Lansky, "Reactive reasoning and planning," in *Proceedings of the Sixth National Conference on Artificial Intelligence, AAAI-87*, 1987, pp. 677–682.
- [16] M. P. Georgeff and F. F. Ingrand, "Managing Deliberation and Reasoning in Real-Time Systems," in *In Proceedings of the DARPA Workshop on Innovative Approaches to Planning*, San Diego, California, 1990.

The 3rd International Workshop on Multi-Agent Systems and Simulation (MAS&S): Towards an Integration of Agent-Oriented Software Engineering and Simulation, MALLOW-MAS&S'09

(Introductory Essay of the Workshop)

Giancarlo Fortino*, Massimo Cossentino[†], Juan Pavón[‡], and Marie-Pierre Gleizes[§]

*Department of Electronics, Informatics and Systems (DEIS)
Università della Calabria
Via P. Bucci cubo 41C, 87036 Arcavacata di Rende (CS), Italy
Email: g.fortino@unical.it

[†]ICAR/CNR
Viale delle Scienze, Ed. 11. 90128 Palermo, Italy
Email: cossentino@pa.icar.cnr.it

[‡]Dep. Ingeniería del Software e Inteligencia Artificial
Universidad Complutense Madrid (Spain)
Ciudad Universitaria s/n, 28040 Madrid, Spain
Email: jpavon@fdi.ucm.es

[§]IRIT, Université Paul Sabatier (France)
118, route de narbonne, 31062 TOULOUSE Cedex 09, France
Email: Marie-Pierre.Gleizes@irit.fr

Abstract

Multi-agent systems (MASs) provide powerful models for representing real-world applications with an appropriate degree of complexity and dynamism. Several industrial experiences have already shown that the use of MAS offers advantages in many different areas such as manufacturing processes, e-Commerce, network management, etc. As MASs in such contexts need to be tested before their deployment and execution, methodologies that support validation through simulation (e.g. discrete-event simulation, agent-based simulation, etc.) of the MAS under development are highly required. In fact, simulation of a MAS cannot only demonstrate that a MAS correctly behaves according to its specifications but can also support the analysis of emergent properties of the MAS under-test. In this context, MAS&S'09 aims at providing a forum for discussing recent advances about the integration of Agent-Based Simulation (ABS) and Agent Oriented Software Engineering (AOSE) methodologies and techniques for the analysis, design, validation and implementation of MASs.

I. INTRODUCTION

MAS&S'09 is at its third edition. The first edition has been jointly held with EUROSIS ISC 2006 (Industrial Simulation Conference), June 5-7, 2006, Palermo, Italy [1]. The second edition has been contextualized in EUROSIS ESM 2007 (European Simulation and Modelling Conference), October 22-24, 2007, St. Julian's, Malta [2]. The best papers of the first edition have also been selected and their extended and revised version published in International Journal of Agent Oriented Software Engineering, 2(1), Inderscience, 2008. MAS&S was conceived for stimulating discussion among researchers and practitioners working on ABS and AOSE, to enable the identification and the definition of methodologies and techniques for integrating them.

Simulation-based agent-oriented methodologies can offer new opportunities to develop more robust and well-tested multi-agent systems as the design of the multi-agent system could be validated, from functional and performance perspectives, before its implementation and subsequently deployment. MAS&S'09 attempts to provide a discussion forum for collecting and comparing diverse experiences on the use and integration of ABS and AOSE with the aim of fostering cross fertilization.

MAS&S'09 is being held as part of MALLOW'09, the second edition of Multi-Agent Logics, Languages, and Organisations (Federated Workshops), 7-11 Sept. Torino, Italy.

This volume contains the ten papers that have been selected by the Program Committee for presentation at the workshop. Each paper received at least two reviews in order to supply the authors with a rich feedback. The paper contributions cover hot topics in the fields of methodologies for ABS, integration of ABS and AOSE methodologies, ABS techniques for testing service oriented systems, and ABS for self-organizing systems.

The best papers will be invited to the open special issue on "Simulation-based Design and Evaluation of Multi-Agent Systems" to be published in Journal of Simulation Modelling Practice and Theory, Elsevier, to appear in 2010.

We would like to thank all authors for their contributions and the members of the Program Committee for the excellent work during the reviewing phase.

Giancarlo Fortino
Massimo Cossentino
Juan Pavón
Marie-Pierre Gleizes
August 7, 2009

II. WORKSHOP COMMITTEES

A. Workshop Organizers

Giancarlo Fortino	Università della Calabria, Italy
Massimo Cossentino	ICAR/CNR, Italy
Juan Pavon	Universidad Complutense Madrid, Spain
Marie-Pierre Gleizes	IRIT - Université Paul Sabatier, France

B. Programme Committee

Jean-Paul Arcangeli	Université Paul Sabatier, France
Carol Bernon	Université Paul Sabatier, France
Paul Davidson	Blekinge Institute of Technology, Sweden
Giuseppe Di Fatta	University of Reading, UK
Alfredo Garro	Università della Calabria, Italy
Paolo Giorgini	Università di Trento, Italy
Samer Hassan	Universidad Complutense Madrid, Spain
Vincent Hilaire	Université de Belfort-MontBéliard, France
Franziska Klügl	University of Würzburg, Germany
Victor Lesser	University of Massachusetts, Amherst, USA
Adolfo López Paredes	University of Valladolid, Spain
Muaz Niazi	Foundation University, Pakistan
Michael J. North	Argonne National Laboratory, USA
Andrea Omicini	Università di Bologna, Italy
Paolo Petta	OFAI, Austria

Luca Sabatucci	ITC-irst, FBK, Italy
Valeria Seidita	Università degli Studi di Palermo, Italy
Wilma Russo	Università della Calabria, Italy
Pietro Terna	Università di Torino, Italy
Erwan Tranvouez	LSIS, France
Adeline Uhrmacher	University of Rostock, Germany
Giuseppe Vizzari	Università di Milano Bicocca, Italy

III. LIST OF PAPERS

- An Industry Use Case: testing SOA systems with MAS simulators
by *Pier-Giovanni Taranti, Carlos Jose Pereira de Lucena and Ricardo Choren*
- Electricity Market (Virtual) Agents
by *Paulo Trigo, Paulo Marques and Helder Coelho*
- Users' Collaboration as a Driver for Reputation System Effectiveness: a Simulation Study
by *Guido Boella, Marco Remondino and Gianluca Tornese*
- Exploiting the easyABMS methodology in the logistics domain
by *Alfredo Garro and Wilma Russo*
- Engineering Development of Agents using the Cooperative Behaviour of their Components
by *Noélie Bonjean, Carole Bernon and Pierre Glize*
- Simulation of Alternative Self-Organization Models for an Adaptive Environment
by *Stefania Bandini, Andrea Bonomi, Giuseppe Vizzari and Vito Acconci*
- Verification & Validation of Agent Based Simulations using the VOMAS (Virtual Overlay Multi-agent System) approach
by *Muaz Niazi, Amir Hussain and Mario Kolberg*
- Agent based modeling and simulation of multi-project scheduling
by *José Alberto Araújo, Juan Pavón, Adolfo López Paredes and Javier Pajares*
- Quick Prototyping and Simulation with the INGENIAS Agent Framework
by *Jorge Gomez-Sanz, Carlos Rodríguez-Fernández and Juan Pavón*
- Multiagent Simulation Model Design Strategies
by *Franziska Klügl*

IV. SPONSORING INSTITUTIONS

Giancarlo Fortino has partially been funded by the Department of Electronics, Informatics and Systems (DEIS), University of Calabria, Italy.

Juan Pavon has partially been funded by the the project *Agent-based Modelling and Simulation of Complex Social Systems (SiCoSSys)*, supported by Spanish Council for Science and Innovation, with grant TIN2008-06464-C03-01, and by the *Programa de Creación y Consolidación de Grupos de Investigación UCM-BSCH GR58/08*.

V. ACKNOWLEDGMENT

The workshop organizers would like to thank the MALLOW organizers (Matteo Baldoni, Cristina Baroglio, and Guido Boella) for their endless support in making possible the organization of the 3rd edition of MAS&S in MALLOW and the chair sponsorships for financial funding.

REFERENCES

- [1] Alessandro Genco, Antonio Gentile and Salvatore Sorce, editors. *Proceedings of the Industrial Simulation Conference (ISC 2006)*, June 5-7, 2006, University of Palermo, Palermo, Italy, 535 pages, ISBN 90-77381-26-0.
- [2] Jaroslav Sklenar, Cyrille Bertelle and Giancarlo Fortino. *Proceedings of the European Simulation and Modeling Conference (ESM 2007)*, October 22-24, 2007, University of Malta, St Julians, Malta, 615 pages, ISBN 978-90-77381-36-6.

An Industry Use Case: testing SOA systems with MAS simulators

Pier-Giovanni Taranti and
Carlos José Pereira de Lucena
PUC-Rio, Rua M. de São Vicente 225
Rio de Janeiro/RJ, Brazil
pier.taranti@les.inf.puc-rio.br, lucena@inf.puc-rio.br

Ricardo Choren
SE/8 - IME, Pça General Tibúrcio, 80
Rio de Janeiro/RJ, Brazil
choren@ime.eb.br

Abstract—System Test in architectures composed of several asynchronous subsystems is a hard task. The simulation of external systems is usually performed in a limited way, considering only test cases provided, one by one. This paper presents a simulator based on MAS, used to test critical operation software. The simulator, with simple architecture and construction, has supported interface and integration testing phases and also is used to obtain performance metrics to validate non-functional requirements.

I. INTRODUCTION

Service Oriented Architecture (SOA) is an integrated software infrastructure and design approach to deliver business functions as shared and reusable services. SOA offers more flexibility and looser coupling being more suitable for internet computing [1]. SOA is often used both in intra-enterprise-integration (e.g. Message Oriented Messaging systems) and in inter-enterprise-integration (e.g. Web services integration) [2]. Indeed, SOA separates functions into distinct units, or services, which developers make accessible over a network in order that users can combine and reuse to build different applications [3]. The system provides the user requirements by orchestrating an activity between two or more services.

SOA systems are often asynchronous thus testing can be very challenging. The continuously increasing size and distribution of SOA systems make the testing task more complex and increase the size of test code [4], [5]. Indeed, the dynamic and adaptive nature of SOA makes most of the existing testing techniques not directly applicable to test services and service-orchestrated systems [6].

Testing of such systems therefore, often goes hand in hand with setting up test systems performing some message exchanges and to analyse the results [2]. Moreover, testers may not have access to the source code of the services provided by the other parties and they may have no control over the executable code, which may run in any computer over the internet. Thus SOA testing can be very time consuming and inefficient, as manual intervention is needed.

Asynchronous supporting systems (simulators) that are able to act both as a provider and as a consumer of SOA application services can be used for testing purposes. Simulators can provide valid and controlled data for running tests, acting as external actors, to verify the expected behavior of orchestrated

systems. The use of simulators is regarded as an effective way to verify and validate SOA applications before these are deployed and executed [7].

To observe the internal behavior of a SOA application, a simulator should send requests to application, receive answers from it and verify if the answers are appropriate, according to pre-defined test cases. Besides, in large SOA applications, not all services may be implemented before testing activities begin. Thus a simulator may also be used to fill in for missing services. Multi-agent distributed simulators provide the flexibility, modularity and scalability desired for simulating complex systems [8]. A multi-agent distributed simulator is a multi-agent system (MAS), i.e. a system composed of interacting software agents [9]. A software agent is a software component that is able to perceive its environment and act according to its design goals [9], [10].

In this paper we present a MAS simulator to perform integration and performance testing in SOA applications before they are deployed into production environment. MAS are capable of simulating systems with large number of heterogeneous entities behaving differently in dynamic situations [9]. Therefore MAS are more suitable for evaluating distributed systems that involve complex interaction between entities, e.g. service orchestration, human interaction with SOA applications. As agents can simulate these interactions, it can be used to test SOA applications since early stages of development, using less resources.

This paper shows an application of the proposed simulator in a maritime system testing. This system was developed by the Brazilian Navy for ship monitoring in international trips in order to improve safety and security at sea. This system was developed using SOA because it should be integrated with the Long Range Information and Tracking (LRIT) system, which was created by the Fifth Amendment to the International Convention for the Safety of Life at Sea (SOLAS).

The rest of the paper is organized as follows. In section 2 we give an overview on testing SOA applications developed using Web services. The LRIT system, focusing on testing issues, is presented in section 3. In section 4 we present the proposed simulator, discussing its use. Section 5 presents some related work. Finally section 6 presents the concluding remarks and points out possible future work directions.

II. TESTING SOA SYSTEMS

In a SOA system, services may be developed by different teams, from different organizations, and the complete application can be orchestrated later. Testing a SOA system presents many issues, including [11]:

- lack of software artifacts (code and structure)
- dealing with incomplete systems (services bind at run time)
- lack of control over components
- lack of trust in information provided by components
- cost of testing

An integration test technique aims at effective observations of the interfaces between parts of software systems through the development and use of executable test scripts. These scripts, implemented using drivers, hard-code the variables and the expected results for each case. This approach helps to detect a considerable amount of faults, but it is not feasible to test all the possible service interdependencies. Changes in a service source code and interface may require changes in the test script. This is not always a simple task since very often there is no stable test environment. Besides, a service may have to rely on other services to properly perform its functionality. Thus adequate testing may have to be postponed to when all service binding actually happens.

When working with SOA, it would be desirable for a test driver to continuously and actively perform case tests, i.e. the driver should be able to keep generating requests to the SOA application. The driver should also be able to process large volume of data to test non-functional requirements such as

availability and performance. To automate the execution of these drivers, service consumers are usually instantiated and executed in application servers. This can be extremely resource and time consuming, and error-prone.

A possible alternative to testing SOA applications is the use of a multi-agent-based simulation system. The pro-active nature of agents can be used to verify the interactions between entities. The agents can generate requests to the existing services and check if their response are appropriate. An agent can also simulate human interaction by generating user-interface related data.

Another advantage of using the agent paradigm to test SOA applications is that an agent can simulate a whole service. For inter-enterprise integration SOA applications, this is very important. Agents can be used to "replace" services that are under development by other organizations, thus simulating the behavior of the whole application, even without all services. The MAS approach has other advantages: the simulator can be evolved along the project in order to support all project phases; agents can be used to perform non-functional requirements testing, and; agents can be used to generate reports.

III. TESTING THE LRIT DATA CENTER

In 2006, the Maritime Safety Committee created the LRIT system to allow the long-range identification and tracking of ships. Tracking of any applicable ship begins with LRIT information being transmitted from the shipborne equipment. The LRIT information transmitted includes the ship's position, time and identification. The LRIT system consists of the following components (fig. 1) [12]:

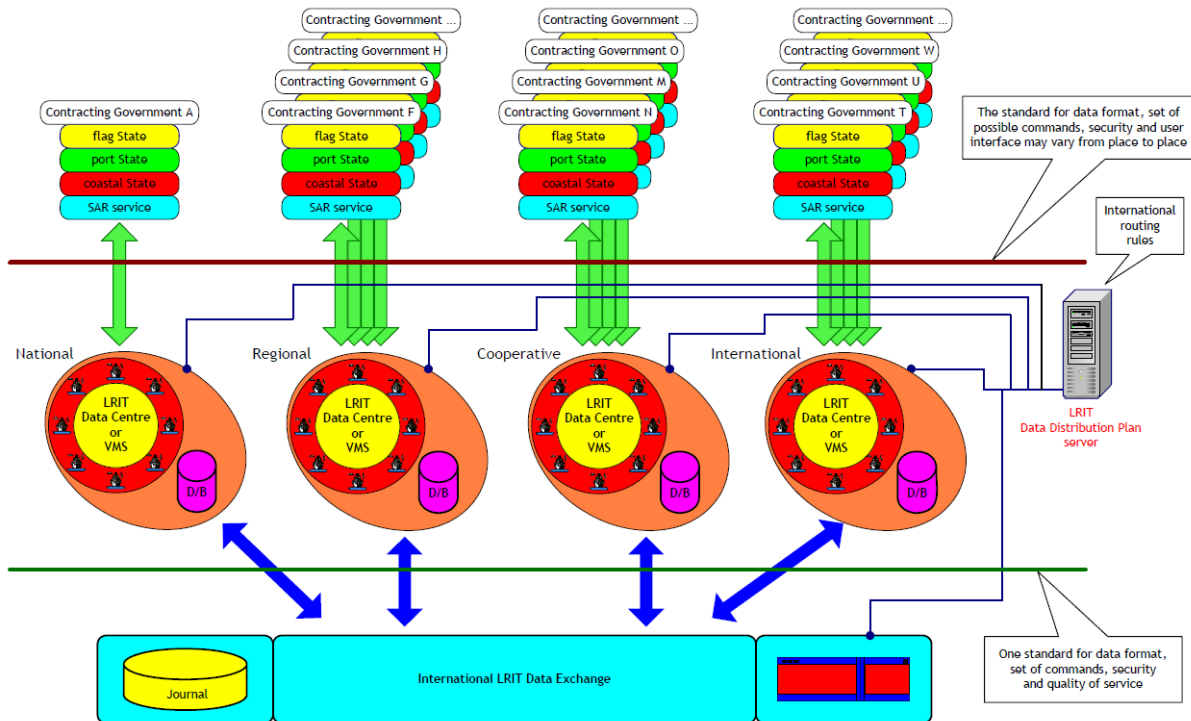


Fig. 1. The LRIT system architecture

Application Service Provider (ASP): the ASP receives the LRIT information from the ship, adds additional information to the LRIT message and passes along the expanded message to its associated Data Center. It provides the functionality required for the programming and communicating of commands to the shipborne equipment.

Data Center (DC): the DC should store all incoming LRIT information from ships instructed by their administrations to transmit LRIT information to that DC. DCs disseminate LRIT information to LRIT data users according to the LRIT Data Distribution Plan (DDP). The DCs process all LRIT messages to and from the International Data Exchange. A DC can provide services to one or more contracting governments.

Data Distribution Plan Server (DDP Server): the DDP contains the information required by the DCs for determining how LRIT information is distributed to the various contracting governments. The DDP contains information such as standing orders from contracting governments and geographical polygons relating to contracting governments' coastal waters.

International Data Exchange (IDE): the IDE routes the message to the appropriate DC based upon the address in the message and the URL/URI in the DDP server. The IDE neither processes nor stores the information contained within LRIT messages.

The Brazilian DC is under development by the Brazilian Navy as a SOA application. To enable the adequate testing of the Brazilian DC services, MAS simulators were used. It is important to mention that the Brazilian DC services were developed independently (and without any coordination) from other LRIT components. Initially, interface simulators were built. They were passive and were used to simulate requests whose results were evaluated manually.

After the first few months of development, a large amount of data needed to be handled at runtime to allow business rules' testing. Business rules' testing involved querying ships that were sailing in real time, changing the frequency of information for individual ships and requests for archived data and all these actions, when performed, needs to consider the data distribution plan for the time when the information was generated. These tests required more advanced functionalities. More specifically, the simulator needed to simulate ships with sailing behavior in given maritime areas and to respond to requests from the Brazilian DC (tested SOA application), following the LRIT rules. Some of these ships needed to have erroneous behavior to verify the data validation of the application.

Thus, the simulator should have an active behavior to simulate the ships that were interacting with the application (e.g. DCs). These ships could change their behavior at runtime, following the communication protocol used by real ships.

IV. THE AGENT-ORIENTED TESTING SIMULATOR

The software agent abstraction is appropriate to handle the problems above agents can represent mobile objects in a georeferenced system. The ShipSim simulator was developed using the maritime domain knowledge acquired when developing

the Dominion [13]. The ShipSim design uses agents to carry all the active behavior expected for an ASP and ships in the system. DC and SimShip interact exchanging SOAP messages, as described below:

SimShip receives SOAP messages from the DC through the ASPSim, which is a passive component that persists incoming SOAP messages in a table of the georeferenced database, used as a blackboard. An AspGateway agent was created to collect the requests sent to ASP from DC. This agent checks the data in blackboard, translates it to ACL [14], and sends it to the agent that simulates the requested ship in the MAS.

Each shipAgent performs the control of compliance with the LRIT communications protocol, responding as expected (or not, if required by a case test). The agent that is simulating a ship starts executing the expected behavior upon receiving a request. The main requests for a ship in the LRIT system are: poll position; change of frequency rate for sending position, and; requests to stop or restart the transmission of positions. Agents simulating ships can be updated to consider the ship's course and speed (fig. 2).

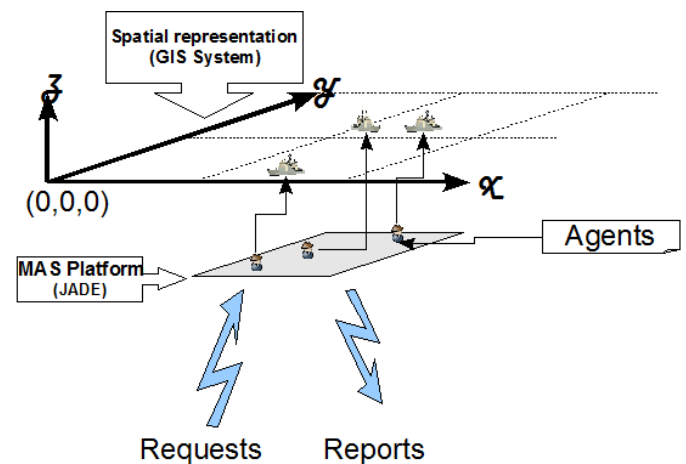


Fig. 2. Ship simulation in the ShipSim

These agents are also capable to control the start/stop and frequency to sending messages for all requestors countries (requests arrives through DC). To send a message to the DC, shipAgents connect directly a DC Web service. This architecture was designed to avoid an ACL to SOAP translation bottleneck. To test if this bottleneck was really overcome, it was necessary to perform tests like maximum number of responded requests per second and number of requests persisted per minute. After the test, the set of reports was used to estimate the maximum merchant fleet that can be supported by the Brazilian DC in the LRIT System. Some designed features for the ShipSim were not implemented at first, like reports and spreadsheets to be used in analyses. However, all features were developed and tested in later iterations.

The architecture used to run the tests is presented in fig. 3. The MAS simulator is the ShipSim, DC is the tested SOA application and the other components are passive interface

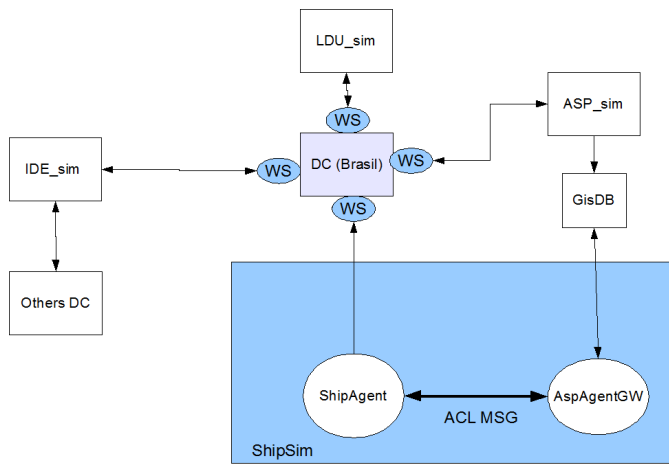


Fig. 3. The testing environment

simulators, implemented using EJB3, running in an application server. These passive simulators basically perform the XML schema validation and allow the exchange of messages using SOAP to the DC (the message is edited in an external application). Since October 2008, IMO had established a complete testing environment, with DCs, IDE and DDP server, however it is not possible to test a DC without data from ships, so the ShipSim is still used to support test activities.

A. Desing and Implementation Details

The methodology presented in Nikraz et al. [15] was used for the testing simulator analyses and design. In this approach agent responsibilities and acquaintances are identified and later mapped to behaviors and communications. ASP and ship services were mapped as responsibilities and acquaintances in the early modeling, and after agent refinement only two types of agent were maintained: the ShipAgent and the ASPAgentGW.

The ASPAgentGW is a transducer agent, who monitors and performs queries on a SQL database, and translate the information to ShipAgent in ACL messages. The ShipAgent aggregate all other test functions: each agent instance simulate an individual ship and is responsible for maintaining a list of all received requests (one for each country) and for answering these requests in correct frequency, informing the current position. The position is calculated using the course and speed of the simulated ship, and those are altered when needed to avoid the exit of a limited area. When sending information, some data is collected, like successful delivery or fail, time to delivery and the sent information.

The testing simulator and the Brazilian DC SOA application were implemented using open source software. Besides, all LRIT standards are open. The simulator was originally developed over the SUSE enterprise version 10.2 operational system, and it is now migrating to version 11.1. The hardware used as runtime environment for the simulator was similar to the LRIT system production environment used by DC: HP servers with two processors quad core and 8 GB of RAM memory.

The ShipSim implementation was done using Java 6, JADE framework [16], Eclipse platform, Java Topology Suite Library (JTS) [17], GIS database (PostgreSQL + PostGIS [18]). The Web services connection was created with the JEE eclipse plug-in, which uses Axis, and supports the SOAP 1.2 protocol. The simulator was modified along the testing to use cryptography with mutual certification over SSL.

The Brazilian DC SOA application was developed with a three layer architecture: the interface layer (for humans and external systems), based in Apache web servers; the application layer, using JEE2 technologies, deployed in a cluster of application servers, and; the persistence layer, using the Hibernate framework. To allow the test execution, the following variables were parameterized:

- square area where agents are created (defined by longitude and latitude limits);
- number of agents (i.e. ships) in simulation;
- time interval between two agents creation (milliseconds);
- interval for each agent sending messages;
- identifiers for agents (i.e. an unique identifier);
- logical port to be used for each build of the simulator (this variable allows the initialization of several simulators simultaneously in the same machine);
- IP address for the tested DC (the project has more than one test environment);
- switch to off/on the cryptography (TSL 1.1 protocol with mutual authentication).

For simpler tests, simulations were executed directly from the Eclipse platform. Whenever it was necessary to place ships in different areas from around the world, a new instance of the simulator (with the specific set of ships) was instantiated. The DC non-functional requirements included information about performance.

The simulator was deployed in different servers, with quad core processors and 8GB of RAM memory. At this environment the simulation was executed with 20.000 agents simultaneously. The simulation RAM consume rate at this situation was 5.6GB. It is important to mention that the initial development effort for the ShipSim required 40 hours of a medium skilled developer, but with experience using JADE and theoretical knowledge about GIS and MAS. This developer has expertise in the LRIT system business rules and maritime environment.

The ShipSim had been modified many times during the LRIT system testing phase. These modifications included simple tasks, mainly because of the JADE architecture. Currently, the ShipSim is used by the Brazilian DC test group, who is responsible for its maintenance. The current ShipSim have four packages: agents; behaviors, objects and support. The simulation is started by the SimulationStarter class, responsible for to charge the jade container and all agents.

B. Sample Testing

This section exemplifies the use of the ShipSim, to show the provided test coverage. At late stages in the development of the LRIT system, seven DCs were incorporated to the system prototype. To allow the Brazilian DC test with them, five

simulators were instantiated to create virtual ships in the South Korea, USA, Canada, Bahamas and Liberia areas (following the polygons published in DDP server).

These simulators generated positions to mobile ships and sent them to the Brazilian DC to verify if the DC could handle a number of messages bigger than the specified in its requirements. All messages were forwarded to the proper DC, and these could make any requests to the ships, because they were active, i.e. they were simulating all behaviors expected for a real ship in the system. Therefore, almost all case tests provided in the IMO Testing Protocol were performed using the ShipSim support.

Some important metrics were obtained from the ShipSim, such as the maximum number of requests the system is capable to receive and persist in a queue per second; the maximum number of requests processed per second; the time the system was unavailable (denial of service), and; the maximum number of ships that the DC can safely support. It is important to stress that all these metrics can be obtained using testing frameworks, but the confidence level is different: using the simulators these metrics can be obtained in a condition near to observed by the real integrated SOA architecture. Figure 4 presents an example of testing simulation with 8000 ships flowing information in the LRIT System.

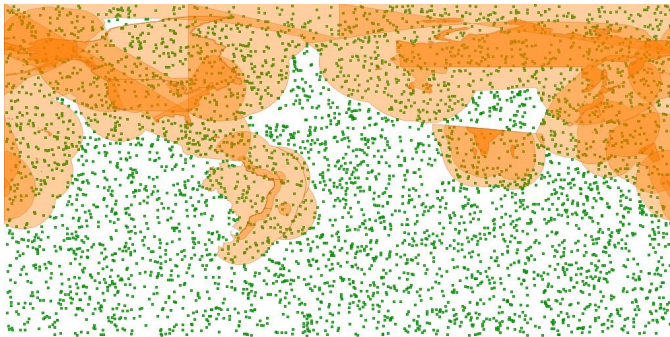


Fig. 4. 8000 ships simulation

V. RELATED WORKS

Canfora and Di Penta [11] present a survey on SOA testing. This work stressed the importance of this activity and it points out some key problems for test execution in SOA applications, especially when a service relies on other services to perform its functionality (late binding between components). None of the works cited in [11] presents a solution for testing a service with well-known interface (WSDL file) had interdependencies with other external (developed by other service providers) services. The proposed simulator approach deals with testing of incomplete systems, using MAS simulation to compose the overall system and then providing a reliable environment for system testing.

Frantzen et al. [19] present a Web services testing technique, using a Model-based testing approach. In this approach, Web services are tested as black-box components. This work points out the need for WSDL descriptions improvement since a

WSDL file only describe the service interface, just like a method signature in Java. The work presents the Symbolic Transition System (STS - a state machine variant) for describing not only the interfaces, but also the executing sequence of a service in order to aid the testing activity. However, the technique focus on Web service testing, i.e. unit testing. It does not consider the specifics of service composition such as late binding, asynchronous communication, unexpected service usage and overall non-functional requirements testing. The proposed simulator approach deals with these issues. The internal logics of the testing stubs are implemented in the agents that can pro-actively interact with the SOA application, thus it does not require a more rigid state machine representation. Testing specifics such as asynchronous communication and unexpected service usage are treated by the simulator agents.

The work presented in [20] shows a discussion about the role of testing and monitoring SOA applications. Testing is considered a preventive activity, to be performed before application delivery (or before using the services). The work states that testing should go beyond monitoring - checking the correctness of the regular service usage - which will be done after the service has been executed. The monitoring could verify Service Level Agreements (SLA) or a specified Quality of Service (QoS). The work considers the need of using both testing and runtime monitoring to improve the confidence in developed SOA applications.

The proposed approach uses software agents to allow SOA application testing and monitoring. Agents can be used to test: a single service; the orchestration of services (i.e. perform integration testing) even when not all services are developed (i.e. an agent acting as a service stub, which is particularly important in inter-enterprise integration applications), and; other qualities (non-functional requirements). In the Brazilian DC system, for instance, the simulators were used to perform quality testing, such as the maximum number of connections, Server Application setting, memory management and code optimizations. This information is usually monitored during system execution, but the agents can verify them during the application development. In this sense, the confidence that the SOA application delivers its functionalities does not rely on post-deployment monitoring, but rather on application verification.

VI. CONCLUSIONS AND FUTURE WORKS

This paper presented an approach for SOA application testing using software agents. The work shows the results of a research on SOA testing that is ongoing for an year. The presented approach intends to allow testing during SOA application development through the use of MAS simulators. The approach deals with some SOA testing issues such as: lack of access to the source code of the parts to be integrated; lack of means of observation on system behavior (mainly due to system incompleteness); lack of control over the services implementation, and; the increasing difficulty and cost of SOA testing.

This paper also presented an experiment that describes a real project. The LRIT system is a distributed system and its components are being developed by different teams. The agent simulators were used to test the overall functionalities and qualities of the Brazilian DC (a LRIT sub-system, developed using the Web service technology by the Brazilian Navy).

The next steps in our work include the specification of a methodology to support agent-based simulator development for SOA application testing. We are seeking to develop a framework to aid the instantiation of simulators. Additionally, we are researching how agent capabilities, such as proactiveness, can help continuous integration activities.

REFERENCES

- [1] C. Lau and A. Ryman, "Developing xml web services with websphere studio application developer," *IBM Systems Journal*, vol. 41, no. 2, pp. 178–197, 2002.
- [2] S. Dustdar and S. Haslinger, "Testing of service oriented architectures: A practical approach," in *Proceedings of 5th Annual International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World (LNCS 3263)*, 2004.
- [3] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2005.
- [4] J. Tian, *Software quality engineering: testing, quality assurance, and quantifiable improvement*. Wiley, 2005.
- [5] J. Z. Gao, H.-S. J. Tsao, and Y. Wu, *Testing and quality assurance for component-based software*. Artech House, 2003.
- [6] G. Canfora and M. Di Penta, "Testing services and service-centric systems: Challenges and opportunities," *IT Professional*, vol. 8, pp. 10–17, 2006.
- [7] W. T. Tsai, Z. Cao, X. Wei, P. Ray, Q. Huang, and X. Sun, "Modeling and simulation in service-oriented software development," *Simulation*, vol. 83, no. 1, pp. 7–32, 2007.
- [8] S. Karnouskos and M. M. J. Tariq, "An agent-based simulation of soa-ready devices," in *Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on*. Cambridge, UK: IEEE, 2008, pp. 330–335.
- [9] M. J. Wooldridge, *An Introduction to MultiAgent Systems*. John Wiley & Sons, Inc., 2002.
- [10] M. Wooldridge and N. Jennings, "Intelligent Agents: Theory and Practice," *The Knowledge Engineering Review*, vol. 10, no. 2, pp. 115–152, 1995.
- [11] G. Canfora and M. Di Penta, "Service-oriented architectures testing: A survey," in *International Summer Schools, ISSSE 2006-2008, Revised Tutorial Lectures (LNCS 5413)*, ser. Software Engineering. Springer Berlin / Heidelberg, 2009, pp. 78–105.
- [12] IMO, "Long range identification and tracking system: Technical documentation (part 1)," International Maritime Organization (IMO), December 2008.
- [13] P. Taranti and R. Choren, "Dominium: an approach to regulate agent societies in dynamic environments," in *Proceedings of the First International Workshop on Agent supported Cooperative Work (ACW) at ICDIM'07*, vol. 2. IEEE, 2007, pp. 811–816.
- [14] FIPA, "Foundation for intelligent physical agents," <http://www.fipa.org/>, 2009.
- [15] M. Nikraz, G. Caire, and P. A. Bahria, "A methodology for the analysis and design of multi-agent systems using jade," *International Journal of Computer Systems Science and Engineering*, vol. 21, no. 2, 2006.
- [16] JADE, "Java agent development framework," <http://jade.tilab.com>, 2009.
- [17] JTS, "Topology suite," <http://www.vividsolutions.com/jts/jtshome.htm>, 2009.
- [18] PostGIS, <http://postgis.refractor.net/>, 2009.
- [19] L. Frantzen, J. Tretmans, and R. de Vries, "Towards model-based testing of web services," in *Proceedings of the International Workshop on Web Services - Modeling and Testing (WS-MaTe 2006)*, 2006.
- [20] G. Canfora and M. Di Penta, "Soa: Testing and self-checking," in *Proceedings of the International Workshop on Web Services ? Modeling and Testing (WS-MaTe 2006)*, 2006.

Electricity Market (Virtual) Agents

Paulo Trigo

LabMAG, GuIAA; DEETC, ISEL
Instituto Superior de Eng. de Lisboa
Portugal

Email: ptrigo@deetc.isel.ipl.pt

Paulo Marques

GuIAA; DEETC, ISEL
Instituto Superior de Eng. de Lisboa
Portugal

Email: 28562@alunos.isel.ipl.pt

Helder Coelho

LabMAG; DI, FCUL
Faculdade de Ciências da Univ. de Lisboa
Portugal

Email: hcoelho@di.fc.ul.pt

Abstract—This paper describes a multi-agent based simulation (MABS) framework to construct an artificial electric power market populated with learning agents. The artificial market, named TEMMAS (The Electricity Market Multi-Agent Simulator), explores the integration of two design constructs: i) the specification of the environmental physical market properties, and ii) the specification of the decision-making (deliberative) and reactive agents. TEMMAS is materialized in an experimental setup involving distinct power generator companies which operate in the market and search for the trading strategies that best exploit their generating units' resources. The experimental results show a coherent market behavior that emerges from the overall simulated environment.

I. INTRODUCTION

The start-up of nation-wide electric markets, along with its recent expansion to intercountry markets, aims at providing competitive electricity service to consumers. The new market-based power industry calls for human decision-making in order to settle the energy assets' trading strategies. The interactions and influences among the market participants are usually described by game theoretic approaches which are based on the determination of equilibrium points to which compare the actual market performance [1], [2]. However, those approaches find it difficult to incorporate the ability of market participants to repeatedly probe markets and adapt their strategies. Usually, the problem of finding the equilibria strategies is relaxed (simplified) both in terms of: i) the human agents' bidding policies, and ii) the technical and economical operation of the power system.

As an alternative to the equilibrium approaches, the multi-agent based simulation (MABS) comes forth as being particularly well fitted to analyze dynamic and adaptive systems with complex interactions among constituents [3], [4].

In this paper we describe a MABS modeling framework that provides constructs for the (human) designer to specify a dynamic environment, its resources, observable properties and its inhabitant decision-making agents. We used the framework to capture the behavior of the electricity market and to build a simulator, named TEMMAS (The Electricity Market Multi-Agent Simulator), which incorporates the operation of several generator company (*GenCo*) operators, each with distinct power generating units (*GenUnit*), and a market operator (*Pool*) which computes the hourly market price (driven by the electricity demand).

TEMMAS agents exhibit bounded rationality, i.e., they make decisions based on local information (partial knowledge) of the system and of other agents while learning and adapting their strategies during a simulation. The TEMMAS purpose is not to explicitly search for equilibrium points, but rather to reveal and assist to understand the complex and aggregate system behaviors that emerge from the interactions of the market agents.

II. THE MABS MODELING FRAMEWORK

We describe the structural MABS constituents by means of two concepts: i) the *environmental entity*, which owns a distinct existence in the real environment, e.g. a resource such as an electricity producer, or a decision-making agent such as a market bidder generator company, and ii) the *environmental property*, which is a measurable aspect of the real environment, e.g. the price of a bid or the demand for electricity. Hence, we define the *environmental entity* set, $\mathcal{E}_T = \{e_1, \dots, e_n\}$, and the *environmental property* set, $\mathcal{E}_Y = \{p_1, \dots, p_m\}$. The whole environment is the union of its entities and properties: $\mathcal{E} = \mathcal{E}_T \cup \mathcal{E}_Y$.

The environmental entities, \mathcal{E}_T , are often clustered in different classes, or types, thus partitioning \mathcal{E}_T into a set, $\mathcal{P}_{\mathcal{E}_T}$, of disjoint subsets, $\mathcal{P}_{\mathcal{E}_T}^i$, each containing entities that belong to the same class. Formally, $\mathcal{P}_{\mathcal{E}_T} = \{\mathcal{P}_{\mathcal{E}_T}^1, \dots, \mathcal{P}_{\mathcal{E}_T}^k\}$ defines a full partition of \mathcal{E}_T , such that $\mathcal{P}_{\mathcal{E}_T}^i \subseteq \mathcal{E}_T$ and $\mathcal{P}_{\mathcal{E}_T} = \cup_{i=1..k} \mathcal{P}_{\mathcal{E}_T}^i$ and $\mathcal{P}_{\mathcal{E}_T}^i \cap \mathcal{P}_{\mathcal{E}_T}^j = \emptyset \forall i \neq j$. The partitioning may be used to distinguish between decision-making agents and available resources, e.g. a company that decides the bidding strategy to pursue or a plant that provides the demanded power.

The environmental properties, \mathcal{E}_Y , can also be clustered, in a similar way as for the environmental entities, thus grouping properties that are related. The partitioning may be used to express distinct categories, e.g. economical, electrical, ecological or social aspects. Another, more technical usage, is to separate constant parameters from dynamic state variables.

The factored state space representation. The state of the simulated environment is implicitly defined by the state of all its environmental entities and properties. We follow a factored representation, that describes the state space as a set, \mathcal{V} , of discrete state variables [5]. Each state variable, $v_i \in \mathcal{V}$, takes on values in its domain $\mathcal{D}(v_i)$ and the global (i.e., over \mathcal{E}) state space, $\mathcal{S} \subseteq \times_{v_i \in \mathcal{V}} \mathcal{D}(v_i)$, is a subset of the Cartesian product of the state variable domains. A state $s \in \mathcal{S}$ is an

assignment of values to the set of state variables \mathcal{V} . We define $f_C, C \subseteq \mathcal{V}$, as a projection such that if s is an assignment to \mathcal{V} , $f_C(s)$ is the assignment of s to C ; we define a *context* c as an assignment to the subset $C \subseteq \mathcal{V}$; the initial state variables of each entity and property are defined, respectively, by the functions $init_{\mathcal{E}_T} : \mathcal{E}_T \rightarrow C$ and $init_{\mathcal{E}_Y} : \mathcal{E}_Y \rightarrow C$.

From environmental entities to resources and agents. The *embodiment* is central in describing the relation between the entities and the environment [6]. Each *environmental entity* can be seen as a body, possibly with the capability to influence the environmental properties. Based on this idea of embodiment, two higher-level concepts (decoupled from the environment, \mathcal{E} , characterization) are introduced: i) *agent*, owing reasoning and decision-making capabilities, and ii) *resource*, without any reasoning capability. Thus, given a set of agents, Υ , we define an association function $embody : \Upsilon \rightarrow \mathcal{E}_T$, which connects an agent to its physical entity. In a similar way, given a set of resources, Φ , we define the mapping function $identity : \Phi \rightarrow \mathcal{E}_Y$. We consider that $|\mathcal{E}| = |\Upsilon| + |\Phi|$, thus each entity is either mapped to an agent or to a resource; there is no third category.

The decision-making approach. Each agent perceives (the market) and acts (sells or buys) and there are two main approaches to develop the reasoning and decision-making capabilities: i) the qualitative mental-state based reasoning, such as the belief-desire-intention (BDI) architecture [7], which is founded on logic theories, and ii) the quantitative, decision-theoretic, evaluation of causal effects, such as the Markov decision process (MDP) support for sequential decision-making in stochastic environments. There are also hybrid approaches that combine the qualitative and quantitative formulations [8], [9].

The qualitative mental-state approaches capture the relation between high level components (e.g. beliefs, desires, intentions) and tend to follow heuristic (or rule-based) decision-making strategies, thus being better fitted to tackle large-scale problems and worst fitted to deal with stochastic environments.

The quantitative decision-theoretic approaches deal with low level components (e.g., primitive actions and immediate rewards) and searches for long-term policies that maximize some utility function, thus being worst fitted to tackle large-scale problems and better fitted to deal with stochastic environments.

The electric power market is a stochastic environment and we currently formulate medium-scale problems that can fit a decision-theoretic agent model. Therefore, TEMMAS adaptive agents (e.g., market bidders) follow a MDP based approach and resort to experience (sampled sequences of states, actions and rewards from simulated interaction) to search for optimal, or near-optimal, policies using reinforcement learning methods such as Q-learning [10] or SARSA [11].

III. TEMMAS DESIGN

Within the current design model of TEMMAS the electricity asset is traded through a spot market (no bilateral agreements), which is operated via a *Pool* institutional power entity. Each

generator company, *GenCo*, submits (to *Pool*) how much energy, each of its generating unit, $GenUnit_{GenCo}$, is willing to produce and at what price. Thus, we have: i) the power supply system comprises a set, \mathcal{E}_{GenCo} , of generator companies, ii) each generator company, *GenCo*, contains its own set, $\mathcal{E}_{GenUnit_{GenCo}}$, of generating units, iii) each generating unit, $GenUnit_{GenCo}$, of a *GenCo*, has constant marginal costs, and iv) the market operator, *Pool*, trades all the *GenCos*' submitted energy.

The bidding procedure conforms to the so-called "block bids" approach [12], where a block represents a quantity of energy being bided for a certain price; also, *GenCos* are not allowed to bid higher than a predefined price ceiling. Thus, the market supply essential measurable aspects are the energy price, quantity and production cost. The consumer side of the market is mainly described by the quantity of demanded energy; we assume that there is no price elasticity of demand (i.e., no demand-side market bidding).

Therefore, we have: $\mathcal{E}_T = \{Pool\} \cup \mathcal{E}_{GenCo} \cup_{g \in \mathcal{E}_{GenCo}} \mathcal{E}_{GenUnit_g}$ where $\mathcal{E}_Y = \{quantity, price, productionCost\}$. The *quantity* refers both to the supply and demand sides of the market. The *price* refers both to the supply bided values and to the market settled (by *Pool*) value.

The \mathcal{E}_{GenCo} contains the decision-making agents. The *Pool* is a reactive agent that always applies the same predefined auction rules in order to determine the market price and hence the block bids that clear the market. Each $\mathcal{E}_{GenUnit_{GenCo}}$ represents the *GenCo*'s set of available resources.

The resources' specification. Each generating unit, $GenUnit_{GenCo}$, defines its marginal costs and constructs the block bids according to the strategy indicated by its generator company, *GenCo*. Each $GenUnit_{GenCo}$ calculates its marginal costs according to, either the "WithHeatRate" [13]) or the "WithCO₂" [14] formulation.

The "WithHeatRate" formulation estimates the marginal cost, MC , by combining the variable operations and maintenance costs, $vO\&M$, the number of heat rate intervals, $nPat$, each interval's capacity, cap_i and the corresponding heat rate value, hr_i , and the price of the fuel, $fPrice$, being used; the marginal cost for a given $i \in [1, nPat]$ interval is given by,

$$MC_{i+1} = vO\&M + \frac{(cap_{i+1} \times hr_{i+1}) - (cap_i \times hr_i)}{blockCap_{i+1}} \times fPrice \quad (1)$$

where each block's capacity is given by: $blockCap_{i+1} = cap_{i+1} - cap_i$.

The "WithCO₂" marginal cost, MC , combines the variable operations and maintenance costs, $vO\&M$, the price of the fuel, $fPrice$, the CO₂ cost, CO_2cost , and the unit's productivity, η , through the expression,

$$MC = \frac{fPrice}{\eta} \times K + CO_2cost + vO\&M \quad (2)$$

where K is a fuel-dependent constant factor, and CO_2cost is given by,

$$CO_2cost = CO_2price \times \frac{CO_2emit}{\eta} \times K \quad (3)$$

where CO_2emit is the CO_2 fuel's emissions. Here all blocks have the same capacity; given a unit's maximum capacity, $maxCap$, and a number of blocks, $nBlocks$, to sell, each block's capacity is given by: $blockCap = \frac{maxCap}{nBlocks}$.

The decision-making strategies. Each generator company defines the bidding strategy for each of its generating units. We designed two types of strategies: a) the *basic-adjustment*, that chooses among a set of basic rigid options, and b) the *heuristic-adjustment*, that selects and follows a predefined well-known heuristic. There are several *basic-adjustment* strategies already defined in TEMMAS. Here we outline seven of those strategies, $sttg_i$ where $i \in \{1, \dots, 7\}$, available for a *GenCo* to apply: i) $sttg_1$, bid according to the marginal production cost of each $GenUnit_{GenCo}$ (follow heat rate curves, e.g., cf. tables II and III), ii) $sttg_2$, make a "small" increment in the prices of all the previous-day's block bids, iii) $sttg_3$, similar to $sttg_2$, but makes a "large" increment, iv) $sttg_4$, make a "small" decrement in the prices of all the previous-day's block bids, v) $sttg_5$, similar to $sttg_4$, but makes a "large" decrement, vi) $sttg_6$, hold the prices of all previous-day's block bids, vii) $sttg_7$ set the price to zero. There are two *heuristic-adjustment* defined strategies: a) the "Fixed Increment Price Probing" (FIPP) that uses a percentage to increment the price of last day's transacted energy blocks and to decrement the non-transacted blocks, and b) "Physical Withholding based on System Reserve" (PWSR) that reduces the block's capacity, as to decrement the next day's estimated system reserve (difference between total capacity and total demand), and then bids the remaining energy at the maximum market price.

The agents' decision process. The above strategies correspond to the *GenCo* agent's primary actions. The *GenCo* has a set, $\mathcal{E}_{GenUnit_{GenCo}}$, of generating units and, at each decision-epoch, it decides the strategy to apply to each generating unit, thus choosing a vector of strategies, $sttg$, where the i^{th} vector's component refers to the $GenUnit_{GenCo}^i$ generating unit; thus, its action space is given by: $\mathcal{A} = \times_{i=1}^{|\mathcal{E}_{GenUnit_{GenCo}}|} \{sttg_1, \dots, sttg_7\}_i \cup \{FIPP, PWSR\}$. The *GenCo*'s perceived market share, $mShare$, is used to characterize the agent internal memory so its state space is given by $mShare \in [0..100]$. Each *GenCo* is a MDP decision-making agent such that the decision process *period* represents a daily market. At each decision-epoch each agent computes its daily profit (that is regarded as an internal reward function) and the *Pool* agent receives all the *GenCos*'s block bids for the 24 daily hours and settles the hourly market price by matching offers in a classic supply and demand equilibrium price (we assume a hourly constant demand).

TEMMAS architecture and construction. The TEMMAS agents along with the major inter-agent communication paths are represented in the bottom region of Figure 1; the top region represents the user interface that enables to specify the each of the resources' and agents' configurable parameters. The implementation of the TEMMAS architecture followed the INGENIAS [15] methodology and used its supporting

development platform. Figure 2 presents the general "agent's perspective", where the tasks and the goals are clustered into individual and social perspectives. Figure 3 gives additional detail on the construction of tasks and goals using INGENIAS.

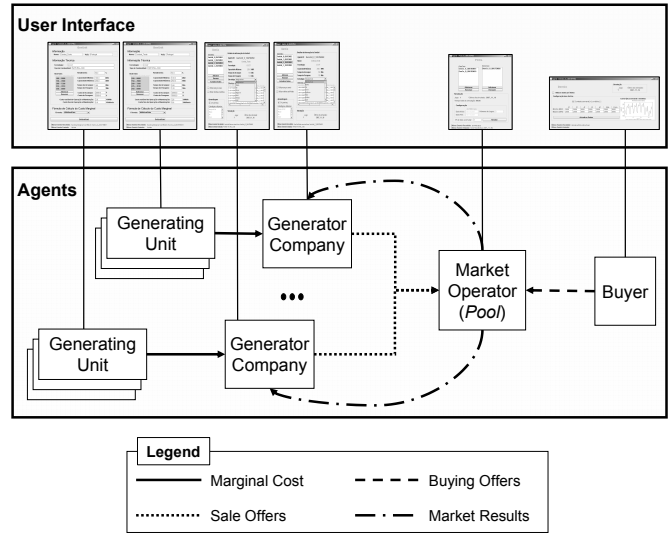


Fig. 1. The TEMMAS architecture and the configurable parameters.

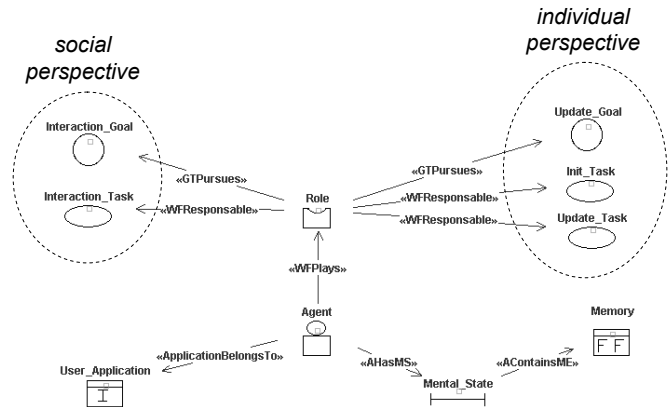


Fig. 2. TEMMAS agent's view using INGENIAS framework.

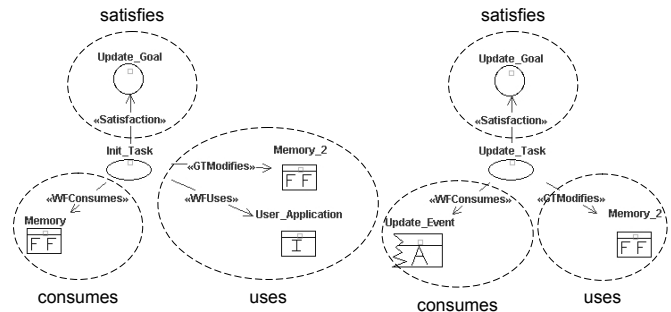


Fig. 3. TEMMAS tasks and goals specification using INGENIAS framework.

IV. TEMMAS ILLUSTRATIVE SETUP

We used TEMMAS to build a specific electric market simulation model. We picked the inspiration from the Iberian Electricity Market (MIBEL – “Mercado Ibérico de Electricidade”) with Portuguese (e.g., EDP - “Elevtrividade de Portugal”, “Turbogás”, “Tejo Energia”) and Spanish (e.g., “Endesa”, “Iberdrola”, “Union Fenosa”, “Hidro Cantábrico”, “Viesgo”, “Bas Natural”, “Elcogás”) generator companies. Regarding the total electricity capacity installed the Iberian market is composed of a major player (Spain) and a minor player (Portugal). Our experiments exploit the combined market behavior of a major and a minor electricity market players.

We abstracted intra-nation market details and modeled each country as a single generator company (with several generating units). Figure 4 uses INGENIAS notation to depict the hierarchical structure of the electricity market; the *Pool* (OMEL – “Operador do Mercado Ibérico de Electricidade”) settles the market price (and coupled bids) after the bids submitted by each *GenCo* (PT – “Portugal” and ES – “Spain”) according to a strategy that depends on the marginal production costs of each *GenUnit*.

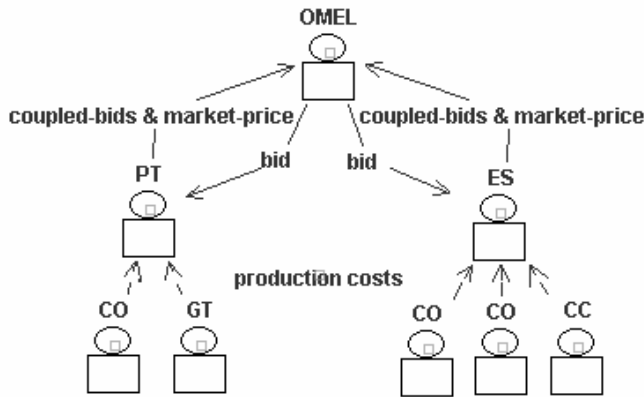


Fig. 4. An illustrative TEMMAS formulation (using INGENIAS notation).

We considered three types of generating units: i) one base load coal plant, CO, ii) one combined cycle plant, CC, to cover intermediate load, and iii) one gas turbine, GT, peaking unit. Table I shows the essential properties of each plant type and tables II and III shows the heat rate curves used to define the bidding blocks. The marginal cost was computed using expression (1); the bidding block’s quantity is the capacity increment, e.g. for CO, the 11.9 marginal cost bidding block’s quantity is 350 – 250 = 100 MW (cf. Table II, CO, top lines 2 and 1).

V. EXPERIMENTS AND RESULTS

Our experiments have two main purposes: i) illustrate the TEMMAS functionality, and ii) analyze the agents’ resulting behavior, e.g. the learnt bidding policies, in light of the market specific dynamics.

We designed three experimental scenarios and Table IV shows the *GenCo*’s name along with its production capacity,

TABLE I
PROPERTIES OF GENERATING UNITS; THE UNITS’ TYPES ARE COAL (CO), COMBINED CYCLE (CC) AND GAS TURBINE (GT); THE O&M INDICATES “OPERATION AND MAINTENANCE” COST.

Property	unit	Type of generating unit		
		CO	CC	GT
Fuel	—	Coal (BIT)	Nat. Gas	Nat. Gas
Capacity	MW	500	250	125
Fuel price	€/MMBtu	1.5	5	5
Variable O&M	€/MWh	1.75	2.8	8

TABLE II
CO AND CC UNIT’S CAPACITY BLOCK (MW) AND HEAT RATE (BTU/KWH) AND THE CORRESPONDING MARGINAL COST (€/MWH).

CO generating unit			CC generating unit		
Cap.	Heat rate	Marg. cost	Cap.	Heat rate	Marg. cost
250	12000	—	100	9000	—
350	10500	11.9	150	7800	29.8
400	10080	12.5	200	7200	29.8
450	9770	12.7	225	7010	30.3
500	9550	13.1	250	6880	31.4

TABLE III
GT UNIT’S CAPACITY BLOCK (MW) AND HEAT RATE (BTU/KWH) AND THE CORRESPONDING MARGINAL COST (€/MWH).

GT generating unit		
Cap.	Heat rate	Marg. cost
50	14000	—
100	10600	44.0
110	10330	46.2
120	10150	48.9
125	10100	52.5

computed according to the respective *GenUnits* (cf. Table I). The “active” suffix (cf. Table IV, *name* column) means that the *GenCo* searches for its *GenUnits* best bidding strategies; i.e. “active” is a policy learning agent.

TABLE IV
THE EXPERIMENT’S *GenCos* AND *GenUnits*.

Exp.	<i>GenCo</i>		<i>GenUnits</i>
	<i>name</i>	Prod. Capac.	
#1	<i>GenCo_active</i>	875	CO & CC & GT
#2	<i>GenCo_major</i>	2000	2×CO & 4×CC
	<i>GenCo_minor&active</i>	875	3×CC & 1×GT
#3	<i>GenCo_major&active</i>	2000	2×CO & 4×CC
	<i>GenCo_minor&active</i>	875	3×CC & 1×GT

Experiment #1. The experiment sets a constant, 600 MW, hourly demand for electricity. Figure 5 shows the *GenCo_active* process of learning the bidding policy that gives the highest long-term profit. We used Q-learning, with an ϵ -greedy exploration strategy, which picks a random action with probability ϵ and behaves greedily otherwise (i.e., picks

the action with the highest estimated action value); we defined $\epsilon = 0.2$. The learning factor rate of Q-learning was defined as $\alpha = 0.01$ and the discount factor (which measures the present value of future rewards) was set to $\gamma = 0.5$. Figure 6 shows the bid blocks that cleared the market (at the first hour of last simulated day). As there is no market competition the cheapest, CO, bids zero, the GT sets the market price (to its ceiling) and the most expensive 200 MW are distributed among the most expensive *GenUnits* (CC, GT). Therefore, the *GenCo_active* agent found, for each perceived market share, $mShare$, the best strategy, $sttg$, to bid its *GenUnits*' energy blocks.

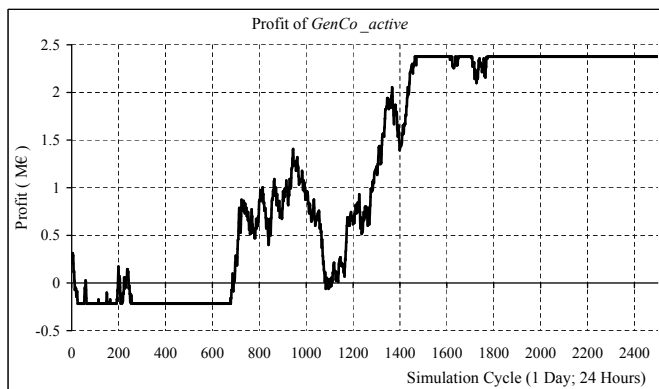


Fig. 5. The process of learning a bid policy to maximize profit. [Exp. #1]

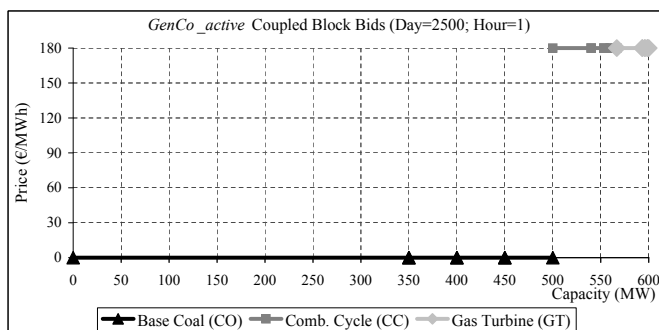


Fig. 6. The bid policy that maximizes profit (price ceiling is 180). [Exp. #1]

Experiment #2. The experiment sets a constant, 2000 MW, hourly demand for electricity. Figure 7 shows the market share evolution while *GenCo_minor&active* learns to play in the market with *GenCo_major*, which is a larger company with a fixed strategy: “bid each block 5€ higher than its marginal cost”. We see that *GenCo_minor&active* gets around 18% (75 – 57) of market from *GenCo_major*. To earn that market the *GenCo_minor&active* learnt to lower its prices in order to exploit the “5€ space” offered by *GenCo_major* fixed strategy.

Experiment #3. In this experiment both *GenCos* are “active”; the remaining is the same as in experiment #2. Figure 8 shows the market share oscillation while each company reacts to the other’s strategy to win the market. Despite the

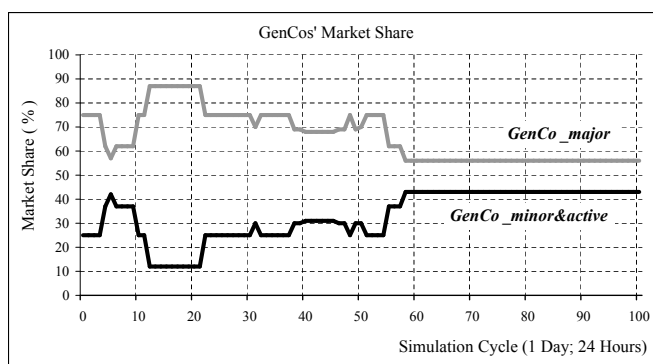


Fig. 7. Market share evolution induced by *GenCo_minor&active*. [Exp. #2]

competition each company learns to secure its own fringe of the market.

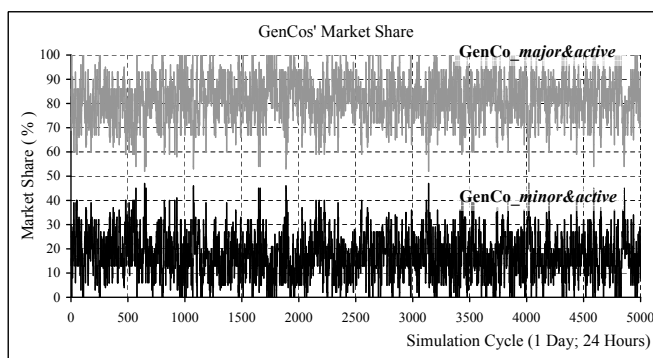


Fig. 8. Market share evolution induced by both *GenCos*. [Exp. #3]

VI. CONCLUSIONS AND FUTURE WORK

This paper describes our preliminary work in the construction of a MABS framework to analyze the macro-scale dynamics of the electric power market. Although both research fields (MABS and market simulation) achieved considerable progress there is a lack of cross-cutting approaches. We used the proposed MABS framework to support our preliminary work in the construction of the TEMMAS agent-based electricity market simulator.

Hence, our contribution is two folded: i) a comprehensive formulation of MABS, including the simulated environment and the inhabiting decision-making and learning agents, and ii) a simulation model (TEMMAS) of the electric power market framed in the proposed formulation.

Our initial results reveal an emerging and coherent market behavior, thus inciting us to further extend the experimental setup with additional bidding strategies and to incorporate specific market rules, such as congestion management and pricing regulation mechanisms.

REFERENCES

- [1] Berry, C., Hobbs, B., Meroney, W., O'Neill, R., Jr, W.S.: Understanding how market power can arise in network competition: a game theoretic approach. *Utilities Policy* 8(3) (September 1999) 139–158

- [2] Gabriel, S., Zhuang, J., Kiet, S.: A Nash-Cournot model for the north american natural gas market. In: Proceedings of the 6th IAEE European Conference: Modelling in Energy Economics and Policy. (2–3 September 2004)
- [3] Schuster, S., Gilbert, N.: Simulating online business models. In: Proceedings of the 5th Workshop on Agent-Based Simulation (ABS-04). (May 3–5 2004) 55–61
- [4] Helleboogh, A., Vizzari, G., Uhrmacher, A., Michel, F.: Modeling dynamic environments in multi-agent simulation. *JAAMAS* **14**(1) (2007) 87–116
- [5] Boutilier, C., Dearden, R., Goldszmidt, M.: Exploiting structure in policy construction. In: Proceedings of the IJCAI-95. (1995) 1104–1111
- [6] Clark, A.: Being there: putting brain, body, and world together again. MIT (1998)
- [7] Rao, A., Georgeff, M.: BDI agents: From theory to practice. In: Proceedings of the First International Conference on Multiagent Systems, S (1995) 312–319
- [8] Simari, G., Parsons, S.: On the relationship between MDPs and the BDI architecture. In: Proceedings of the AAMAS-06. (May 8–12 2006) 1041–1048
- [9] Trigo, P., Coelho, H.: Decision making with hybrid models: the case of collective and individual motivations. *International Journal of Reasoning Based Intelligent Systems (IJRIS)*; Inderscience Publishers (2009)
- [10] Watkins, C., Dayan, P.: Q-learning. *Mach. Learning* **8** (1992) 279–292
- [11] Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT P. (1998)
- [12] : OMIP - The Iberian Electricity Market Operator. online: ‘<http://www.omip.pt>’
- [13] Botterud, A., Thimmapuram, P., Yamakado, M.: Simulating GenCo bidding strategies in electricity markets with an agent-based model. In: Proceedings of the 7th Annual IAEE European Energy Conference (IAEE-05). (August 28–30 2005)
- [14] Sousa, J., Lagarto, J.: How market players adjusted their strategic behaviour taking into account the CO2 emission costs - an application to the spanish electricity market. In: Proceedings of the 4th International Conference on the European Electricity Market (EEM-07), Cracow, Poland (May 23–27 2007)
- [15] Gómez-Sanz, J., Fuentes-Fernández, R., Pavón, J., García-Magariño, I.: INGENIAS development kit: a visual multi-agent system development environment (BEST ACADEMIC DEMO OF AAMAS'08). In: Proceedings of the Seventh AAMAS, Estoril, Portugal (May 12-16 2008) 1675–1676

Users' Collaboration as a Driver for Reputation System Effectiveness: a Simulation Study

Guido Boella and Marco Remondino
Department of Computer Science, University of Turin
boella@di.unito.it , remond@di.unito.it

Gianluca Tornese (for the implementation)
gianluca.tornese@libero.it

Abstract

Reputation management is about evaluating an agent's actions and other agents' opinions about those actions, reporting on those actions and opinions, and reacting to that report thus creating a feedback loop. This social mechanism has been successfully used, through Reputation Management Systems (RMSs) to classify agents within normative systems. Most RMSs rely on the feedbacks given by the member of the social network in which the RMS itself operates. In this way, the reputation index can be seen as an endogenous and self produced indicator, created by the users for the users' benefit. This implies that users' participation and collaboration is a key factor for the effectiveness a RMS. In this work the above factor is explored by means of an agent based simulation, and is tested on a P2P network for file sharing.

1. Introduction

In everyday's life, when a choice subject to limited resources (like for instance money, time, and so on) must be done, due to the overwhelming number of possibilities that people have to choose from, something is needed to help them make choices. People often follow the advice of others when it comes to which products to buy, which movies to watch, which music to listen, which websites to visit, and so on. This is a social attitude that uses others' experience. They base their judgments of whether or not to follow this advice partially upon the other person's reputation in helping to find reliable and useful information, even with all the noise.

Using and building upon early collaboration filtering techniques, reputation management software gather ratings for people, companies, and information sources. Since this is a distributed way of computing

reputation, it is implicitly founded on two main assumptions:

- 1) The correctness of shared information
- 2) The participation of users to the system

While the negation of the first could be considered as an attack to the system itself, performed by users trying to crash it, and its occurrence is quite rare, the second factor is often underestimated, when designing a collaborative RMS. Users without a vision of the macro level often use the system, but simply forget to collaborate, since this seems to cause a waste of time.

The purpose of the present work is to give a qualitative and, when possible, quantitative evaluation of the collaborative factor in RMSs, by means of an empirical analysis conducted via an agent based simulation. Thus, the main research question is: what's the effectiveness of a RMS, when changing the collaboration rate coming from the involved users?

In order to answer this question, in the paper an agent based model is introduced, representing a peer-to-peer (P2P) network for file sharing. A basic RMS is applied to the system, in order to help users to choose the best peers to download from. In fact, some of the peers are malicious, and they try to exploit the way in which the P2P system rewards users for sharing files, by uploading inauthentic resources when they do not own the real ones. The model is described in detail and the results are evaluated through a multi-run *coeteris paribus* technique, in which only one setting is changed at a time. In particular, the most important parameters which will be compared, to evaluate the effectiveness of the RMS are: verification of the files, performed by the users and negative payoff, given in case a resource is reported as being inauthentic. The verification of the files, i.e. users' collaboration, is an exogenous factor for the RMS, while the negative

payoff is an endogenous and thus directly controllable factor, from the point of view of a RMS's designer.

The P2P framework has been chosen since there are many works focusing on the reputation as a system to overcome the issue of inauthentic files, but, when evaluating the effectiveness of the system, the authors [1] usually refer to idealized situations, in which users always verify the files for authenticity, as soon as they start a download. This is obviously not the case in the real world: first of all, most resources require to be at least partially owned, in order to be checked. Besides, some users could simply decide not to check them for long time. Even worse, other users could simply forget about a downloaded resource and never check it. Last but not least, other users might verify it, but simply not report anything, if it's not authentic.

2. Reputation and P2P Systems

Since uploading bandwidth is a limited resource and the download priority queues are based on a uploading-credit system to reward the most collaborative peers on the network, some malicious users create inauthentic files, just to have something to share, thus obtaining credits, without being penalized for their behavior. To balance this, RMSs have been introduced, which dynamically assign to the users a reputation value, considered in the decision to download files from them or not. RMSs are proven, via simulation, to make P2P networks safe from attacks by malicious peers, even when forming coalitions. In networks of millions of peers attacks are less frequent, but users still have a benefit from sharing inauthentic files. It's not clear if RMSs can be effective against this selfish widespread misbehavior, since they make several ideal assumptions about the behavior of peers who have to verify files to discover inauthentic ones. This operation is assumed to be automatic and with no costs. Moreover, since the files are usually shared before downloading is completed, peers downloading inauthentic files unwillingly spread them if they are not cooperative enough to verify their download as soon as possible. In the present work, the creation and spreading of inauthentic files is not considered as an attack, but as a way in which some agents try to raise their credits, while not possessing the real resource that's being searched by others. A basic RMSs is introduced, acting as a positive or negative reward for the users and human factor behind the RMSs is considered, in the form of costs and benefits of verifying files. Most approaches, most notably EigenTrust [2], assume that verification is made automatically upon the start of download of the file. By looking as we do at the collaboration factor in dealing

with RMSs, we can question their real applicability, an issue which remains unanswered in the simulation based tests made by the authors. To provide an answer to this question it is necessary to build a simulation tool which aims at a more accurate modeling of the users' behavior rather than at modeling the reputation system in detail.

3. Model Framework

We assume a simple idealized model of reputation, since the objective is not to prove the effectiveness of a particular algorithm but to study the effect of users' behavior on a reputation system. We use a centralized system which assumes the correctness of information provided by users, e.g., it is not possible to give an evaluation of a user with whom there was no interaction. When verifying a file, the agents give a negative payoff to the agent uploading it, in case it's inauthentic. In turn, the system will spread it to the agents (if any) who uploaded it to the sender. There are two reputation thresholds: the first and higher one, under which it's impossible to ask for resources to other agents, the second, lower than the other, which makes it impossible even to share the owned files. This guarantees that an agents that falls under the first one (because she shared too many inauthentic files), can still regain credits by sharing authentic ones and come back over the first threshold. On the contrary, if she continues sharing inauthentic files, she will fall also under the second threshold, being de facto excluded from the network, still being a working link from and to other agents. The agents are randomly connected on a graph and feature the following parameters: Unique ID, Reputation value, set of neighbors, set of owned resources, set of goals (resources), set of resources being downloaded, set of suppliers (by resource). At each time step, agents reply to requests for download, perform requests (according to their goals) or verify files. While an upload is performed – if possible - each time another agent makes a request, requesting a resource and verification are performed in alternative. Verification ratio is a parameter for the simulation and acts stochastically on agents' behavior. All agents belong to two disjoint classes: malicious agents and loyal ones. They have different behaviors concerning uploading, while feature the same behavior about downloading and verification: malicious agents are simply agents who exploit for selfishness the weaknesses of the system, by always uploading inauthentic files if they don't own the authentic ones. Loyal agents, on the contrary, only upload a resource if they own it. A number of resources are introduced in the system at the beginning of the simulation,

representing both the owned objects and the agents' goals. For coherence, an owned resource can't be a goal, for the same agent. The distribution of the resource is stochastic. During the simulation, other resources (and corresponding goals) are stochastically distributed among the agents. Each agent (metaphorically, the P2P client) keeps track of the providers, and this information is preserved also after the download is finished.

To test the limits and effectiveness of a reputation mechanism under different user behaviors an agent based simulation of a P2P network is used as methodology, employing reactive agents to model the users; these have a deterministic behavior based on the class they belong to (malicious or loyal) and a stochastic idealized behavior about verifying policy. Their use shows how the system works at an aggregate level. However, reactive agents can also be regarded as a limit for our approach, since real users have a flexible behavior and adapt themselves to what they observe. We built a model which is less idealized about the verifying factor, but it's still rigid when considering the agents' behavior about sending out inauthentic files. That's why we envision the necessity to employ cognitive agents based on reinforcement learning techniques. Though, reactive agents can also be a key point, in the sense that they allow the results to be easily readable and comparable among them, while the use of cognitive agents would have moved the focus from the evaluation of collaborative factor to that of real users' behavior when facing a RMS, which is very interesting, but beyond the purpose of the present work. In future works, this paradigm for agents will be considered.

The model is written in pure Java and does not make use of any agent development environment.

4. Model Specifications and Parameters

The P2P network is modeled as an undirected and non-reflexive graph. Each node is an agent, representing a P2P user. Agents are reactive: their behavior is thus determined a priori, and the strategies are the result of the stimuli coming from the environment and of the condition-action rules. Their behavior is illustrated in next section. Formally the multi agent system is defined as $MAS = \langle Ag; Rel \rangle$, with Ag set of nodes and Rel set of edges. Each edge among two nodes is a link among the agents and is indicated by the tuple $\langle ai; aj \rangle$ with ai and aj belonging to Ag . Each agent features the following internal parameters:

- Unique ID (identifier),

- Reputation value (or credits) $N(ai)$,
- Set of agent's neighbors $RP(ai)$,
- Set of owned resources $RO(ai)$,
- Set of goals (resource identifiers) $RD(ai)$,
- Set of resources being downloaded $P(ai)$,
- Set of pairs $\langle \text{supplier}; \text{resource} \rangle$.

A resource is a tuple $\langle Name, Authenticity \rangle$, where $Name$ is the resource identifier and $Authenticity$ is a Boolean attribute indicating whether the resource is authentic or not. The agent owning the resource, however, does not have access to this attribute unless he verifies the file.

The resources represent the object being shared on the P2P network. A number of resources are introduced in the system at the beginning of the simulation; they represent both the owned objects and the agents' goals. For coherence, an owned resource can't be a goal, for the same agent. The distribution of the resource is stochastic. During the simulation, other resources are stochastically introduced. In this way, each agent in the system has the same probabilities to own a resource, independently from her inner nature (malicious or loyal). In the same way also the corresponding new goals are distributed to the agents; the difference is that the distribution probability is constrained by its being possessed by an agent. Formally R be the set of all the resources in the system. We have that:

$$RD(ai) \subseteq R, RO(ai) \subseteq R \text{ and } RD(ai) \cap RO(ai) = \emptyset.$$

Each agent in the system features a set of neighbors $N(ai)$, containing all the agents to which she is directly linked in the graph: $N(ai) = \{aj \in Ag \mid \langle ai; aj \rangle \in Rel\}$. This information characterizes the information of each agent about the environment. The implemented protocol is a totally distributed one, so looking for the resource is heavily based on the set of neighbors.

In the real word the shared resources often have big dimensions; after finding the resource, a lot of time is usually required for the complete download. In order to simulate this the set of the "resources being downloaded" (Ris) introduced. These are described as $Ris = \langle resource\ ID, completion, check\ status \rangle$, where ID is the resource identifier, $completion$ is the percentage already downloaded and " $check\ status$ " indicates whether the resource has been checked for authenticity or not. In particular, it can be not yet verified, verified and authentic and verified and inauthentic:

$$check\ status \in \{NOT\ CHECKED; AUTH; INAUTH\}$$

Another information is ID of the provider of a certain resource, identified by $P(ai)$. Each agent keeps track of those which are uploading to him, and this information is preserved also after the download is finished. The real P2P systems allow the same resource to be download in parallel from many providers, to improve

the performance and to split the bandwidth load. This simplification should not affect the aggregate result of the simulation, since the negative payoff would reach more agents instead of just one (so the case with multiple provider is a sub-case of that with a single provider).

4.1. The Reputation Model

In this work we assume a simple idealized model of reputation, since the objective is not to prove the effectiveness of a particular reputation algorithm but to study the effect of users' behavior on a reputation system. We use a centralized system which assumes the correctness of information provided by users, e.g., it is not possible to give an evaluation of a user with whom there was no interaction. The reason is that we focus on the behavior of common agents and not on hackers who attack the system by manipulating the code of the peer application. In the system there are two reputation thresholds: the first and higher one, under which it's impossible to ask for resources to other agents, the second, lower than the other, which makes it impossible even to share the owned files. This guarantees that an agent that falls under the first one (because she shared too many inauthentic files), can still regain credits by sharing authentic ones and come back over the first threshold. On the contrary, if she continues sharing inauthentic files, she will fall also under the second threshold, being de facto excluded from the network, still being a working link from and to other agents.

4.2. The User Model

Peers are reactive agents replying to requests, performing requests or verifying files. While upload is performed each time another agent makes a request, requesting a file and verification are performed (in alternative) when it is the turn of the agent in the simulation. All agents belong to two disjoint classes: malicious agents and loyal agents. The classes have different behaviors concerning uploading, while they have the same behavior concerning downloading and verification: malicious agents are just common agents who exploit for selfishness the weaknesses of the system. When it is the turn of another peer, and he requests a file to the agent, he has to decide whether to comply with the request and to decide how to comply with it.

- The decision to upload a file is based on the reputation of the requester: if it is below the "replying

threshold", the requestee denies the upload (even if the requestee is a malicious agent).

- The "replyTo" method refers to the reply each agent gives when asked for a resource. When the agent is faced with a request he cannot comply but the requester's reputation is above the "replying threshold", if he belongs to the malicious class, he has to decide whether to create and upload an inauthentic file by copying and renaming one of his other resources. The decision is based depending on a parameter. If the resource is owned, she sends it to the requesting agent, after verifying if her reputation is higher than the "replying threshold". Each agent performs at each round of simulation two steps:

1) Performing the downloads in progress. For each resource being downloaded, the agents check if the download is finished. If not, the system checks if the resource is still present in the provider's "sharing pool". In case it's no longer there, the download is stopped and is removed from the list of the "owned resources". Each file is formed by n units; when $2/n$ of the file has been downloaded, then the file gets automatically owned and shared also by the agent that is downloading it.

2) Making new requests to other peers or verifying the authenticity of a file downloaded or in downloading, but not both:

a) When searching for a resource all the agents within a depth of 3 from the requesting one are considered. The list is ordered by reputation. A method is invoked on every agent with a reputation higher than the "requests threshold", until the resource is found or the list reaches the ending point. If the resource is found, it's put in the "downloading list", the goal is cancelled, the supplier is recorded and linked with that specific download in progress and her reputation is increased according to the value defined in the simulation parameters. If no resource is found, the goal is given up.

b) Verification means that a file is previewed and if the content does not correspond to its description or filename, this fact is notified to the reputation system. Verification phase requires that at least one file must be in progress and it must be beyond the $2/n$ threshold described above. An agent has a given probability to verify instead of looking for a new file. In case the agent verifies, a random resource is selected among those "in download" and not checked. If authentic, the turn is over. Otherwise, a "punishment" method is invoked, the resource deleted from the "downloading" and from the

"owned " lists and put among the "goals" once again.

The RMS is based on the "punishment" method which lowers the supplier's reputation, deletes her from the "providers" list in order to avoid cyclic punishment chains, and recursively invokes the "punishment" method on the punished provider. A punishment chain is thus created, reaching the creator of the inauthentic file, and all the aware or unaware agents that contributed in spreading it.

5. Results

The simulation goes on until at least one goal exists and/or a download is still in progress.

In the following table a summary of the most important parameters for the experiments are given:

Parameters	Value
Total number of agents	50
Total number of graph edges	80
Initial reputation (credits) for each agent	50
Percentage of loyal agents	50
Total number of resources at the beginning	50
Threshold to share owned resources	25
Threshold for requesting resources	10
Number of turns for introduction of new resources	1
Number of new resources to introduce	3
Total number of steps	2000

Table 1 – the main parameters

In all the experiments, the other relevant parameters are fixed, while the following ones change:

Parameters	Exp 1	Exp 2	Exp 3	Exp 4	Exp 5	Exp 6
Positive payoff	1	1	1	1	1	1
Negative payoff	3	3	4	8	0	2
Verification percentage	30%	40%	30%	30%	30%	40%

Table 2 – the scenarios

A crucial index, defining the wellbeing of the P2P system, is the ratio among the number of inauthentic resources and the total number of files on the network. The total number is increasing more and more over time, since new resources are introduced iteratively. Another measure collected is the average reputation of loyal and malicious agents at the end of the simulation; in an ideal world, we expect malicious ones to be penalized for their behavior, and loyal ones to be rewarded. The results were obtained by a batch execution mode for the simulation. This executes 50

times the simulation with the same parameters, sampling the inauthentic/total ratio every 50 steps. This is to overcome the sampling effect; many variables in the simulation are stochastic, so this technique gives an high level of confidence for the produced results. In 2000 turns, we have a total of 40 samples. After all the executions are over, the average for each time step is calculated, and represented in a chart. In the same way, the grand average of the average reputations for loyal and malicious agents is calculated, and represented in a bar chart. In figure 1, the chart with the trend of inauthentic/total resources is represented for the results coming from experiments 1, 2, 3, 5 and 6. The results of experiment 4 is discussed later.

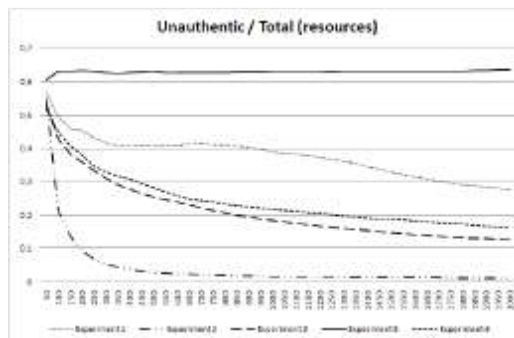


Figure 1 – inauthentic/total ratio

Experiment 5 depicts the worst case: no negative payoff is given: this is the case of a P2P network without a RMS behind it. The ratio initially grows and, at a certain point, it gets constant over time, since new resources are stochastically distributed among all the agents with the same probability. In this way also malicious agents have new resources to share, and they will send out inauthentic files only for those resources they do not own. In the idealized world modeled in this simulation, since agents are 50 malicious and 50 loyal, and since the ones with higher reputation are preferred when asking for a file, it's straightforward that malicious agents' reputation fly away, and that an high percentage of files in the system are inauthentic (about 63%). Experiment 1 shows how a simple RMS, with quite a light punishing factor (3) is already sufficient to lower the percentage of inauthentic files in the network over time. We can see a positive trend, reaching about 28% after 2000 time steps, which is an over 100% improvement compared to the situation in which there was no punishment for inauthentic files. In this experiment the verification percentage is at 30%. This is quite low, since it means that 70% of the files remain unchecked forever (downloaded, but never used). In

order to show how much the human factor can influence the way in which a RMS works, in experiment 2 the verification percentage has been increased up to 40%, leaving the negative payoff still at 3. The result is surprisingly good: the inauthentic/total ratio is dramatically lowered after few turns (less than 10% after 200), reaching less than 1% after 2000 steps. Since 40% of files checked is quite a realistic percentage for a P2P user, this empirically proves that even the simple RMS proposed here dramatically helps in reducing the number of inauthentic files. In order to assign a quantitative weight to the human factor, in experiment 3, the negative payoff is moved from 3 to 4, while bringing back the verification percentage to 30%. Even with a higher punishing factor, the ratio is worse than in experiment 2, meaning that it's preferable to have a higher verification rate, compared to a higher negative payoff. Experiment 6 shows the opposite trend: the negative payoff is lighter (2), but the verification rate is again at 40%, as in experiment 2. The trend is very similar – just a bit worse - to that of experiment 3. In particular, the ratio of inauthentic files, after 2000 turns, is about 16%. At this point, it gets quite interesting to find the “*break even point*” among the punishing factor and the verification rate. After some empirical simulations, we have that, compared with 40% of verification and 3 negative payoff, if now verification is just at 30%, the negative payoff must be set to a whopping value of 8, in order to get a comparable trend in the ratio. This is done in experiment 4 (figure 2): after 2000 turns, there's 1% of inauthentic files with a negative payoff of 3 and a verification percentage of 40%, and about 0.7 with 8 and 30% respectively.

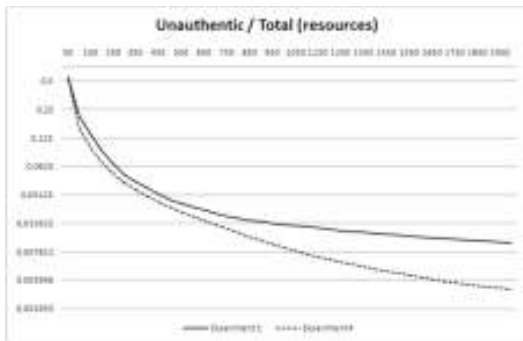


Figure 2 – weighting the collaboration factor

This clearly indicates that collaboration factor (the files verification) is crucial for a RMS to work correctly and give the desired aggregate results (few inauthentic files over a P2P network). In particular, a

slightly higher verification rate (from 30% to 40%) weights about the same of a heavy upgrade of the punishing factor (from 3 to 8). This can be considered as a quantitative result, comparing the exogenous factor (resource verification performed by the users) to the endogenous one (negative payoff).

Besides considering the ratio of inauthentic files moving on a P2P network, it's also crucial to verify that the proposed RMS algorithm could punish the agents that maliciously share inauthentic files, without involving too much unwilling accomplices, which are loyal users that unconsciously spread the files created by the former ones. This is considered by looking at the average reputations, at the end of simulation steps (figure 3).

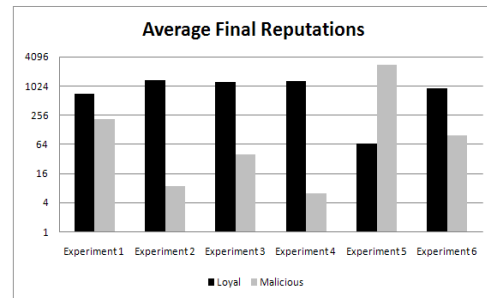


Figure 3 – final average reputations

In the worst case scenario, the malicious agents, that are not punished for producing inauthentic files, always upload the file they are asked for (be it authentic or not). In this way, they soon gain credits, topping the loyal ones. Since in the model the users with a higher reputation are preferred when asking files, this phenomenon soon triggers an explosive effects: loyal agents are marginalized, and never get asked for files. This results in a very low average reputation for loyal agents (around 70 after 2000 turns) and a very high average value for malicious agents (more than 2800) at the same time. In experiment 1 the basic RMS presented here, changes this result; even with a low negative payoff (3) the average reputations after 2000 turns, the results are clear: about 700 for loyal agents and slightly more than 200 for malicious ones. The algorithm preserves loyal agents, while punishing malicious ones. In experiment 2, with a higher verification percentage (human factor), we see a tremendous improvement for the effectiveness of the RMS algorithm. The average reputation for loyal agents, after 2000 steps, reaches almost 1400, while all the malicious agents go under the lower threshold (they can't either download or share resources), with an average reputation of less than 9 points. Experiment 3

explores the scenario in which the users just check 30% of the files they download, but the negative payoff is raised from 3 to 4. The final figure about average reputations is again very good. Loyal agents, after 2000 steps, averagely reach a reputation of over 1200, while malicious ones stay down at about 40. This again proves the proposed RMS system to be quite effective, though, with a low verification rate, not all the malicious agents get under the lower threshold, even if the negative payoff is 4. In experiment 6 the verification percentage is again at the more realistic 40%, while negative payoff is reduced to 2. Even with this low negative payoff, the results are good: most malicious agents fall under the lowest threshold, so they can't share files and they get an average reputation of about 100. Loyal agents behave very well and reach an average reputation of more than 900. Experiment 4 is the one in which we wanted to harshly penalize inauthentic file sharing (negative payoff is set at 8), while leaving an high laxity in the verification percentage (30%). Unlikely what it could have been expected, this setup does not punish too much loyal agents that, unwillingly, spread unchecked inauthentic files. After 2000 turns, all the malicious agents fall under the lowest threshold, and feature an average reputation of less than 7 points, while loyal agents fly at an average of almost 1300 points. The fact that no loyal agent falls under the "point of no return" (the lowest threshold) is probably due to the fact that they do not systematically share inauthentic files, while malicious agents do. Loyal ones just share the inauthentic resources they never check. Malicious agents, on the other side, always send out inauthentic files when asked for a resource they do not own, thus being hardly punished by the RMS, when the negative payoff is more than 3.

6. Whitewashing

A "whitewashing" mode is implemented and selectable before the simulation starts, in order to simulate the real behavior of some P2P users who, realizing that they cannot download anymore (since they have low credits or, in this case, bad reputation), disconnect their client, and then connect again, so to start from the initial pool of credits/reputation. When this mode is active, at the beginning of each turn all the agents that are under a given threshold reset it to the initial value, metaphorically representing the disconnection and reconnection. In experiments 7, 8 and 9 this is tested to see if it affects previous results.

In figure 4, the ratio among inauthentic and total resources is depicted, and in figure 5 the final average

reputation for agents, when whitewashing mode is active.

Even with CBM activated, the results are very similar to those in which this mode is off. They are actually a bit worse when the negative payoff is low (3) and so is the verification percentage (30%): the ratio of inauthentic files in the network is quite high, at about 41% after 2000 turns versus the 27% observed in experiment 1, which had the same parameters, but no CBM. When the verification percentage is increased to 40%, though, things get quite better. Now the ratio of inauthentic files has the same levels as in experiment 2 (less than 1% after 2000 steps). Also with a lower verification percentage (again at 30%), but leaving the negative payoff at 4, the figure is almost identical to the one with the same parameters, but without a CBM. After 2000 turns, the inauthentic files ratio is about 12%.

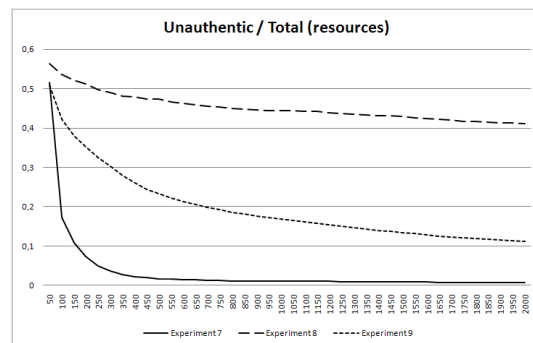


Figure 4 – inauthentic/total ratio in whitewashing mode

The experiments show that malicious agents, even resetting their own reputation after going below the lowest threshold, can't overcome this basic RMS, if they always produce inauthentic files. This happens because, even if they reset their reputation to the initial value, it's still low compared to the one reached by loyal agents; if they shared authentic files, this value would go up in few turns, but since they again start spreading inauthentic files, they almost immediately fall under the thresholds again.

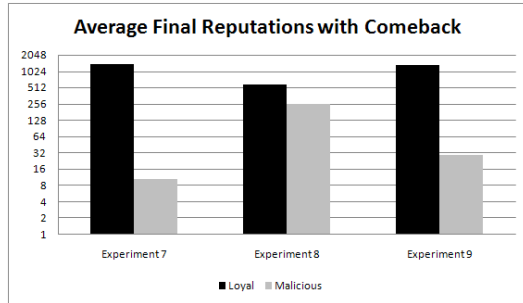


Figure 5 – final average reputations in whitewashing mode

7. Conclusion and Outlook

The main purpose of the work was to show, by means of an empirical analysis based on simulation, how the collaboration coming from the agents in a social system can be a crucial driver for the effectiveness of a RMS.

As a test-bed we considered a P2P network for file sharing and, by an agent based simulation, we show how a basic RMS can be effective to reduce inauthentic files circulating on the network. In order to enhance its performance, though, the collaboration factor, in the form of verifying policy, is crucial: a 33% more in verification results in about thirty times less inauthentic files on the network. While a qualitative analysis of this factor is straightforward for the presented model, we added a quantitative result, trying to weight the exogenous factor (the verification rate) by comparing it to the endogenous one (the negative payoff). We showed that a 33% increase in verification percentage leads to similar results obtained by increasing the negative payoff of 66%. Again, the collaboration factor proves to be crucial for the RMS to work efficiently.

While the provided results are encouraging, the model is not yet realistic under certain aspects. The weakest part is not the simplicity of the RMS algorithm or of the representation of the P2P network, rather the deterministic (reactive) behavior of the agents: the agents involved are too naive to represent real users. In particular, potentially malicious users try to exploit the weaker points of the system, by changing their behavior according to what they observe, like satisfaction of their own goals. It's very unlikely that users, when realizing not to download at the same rate as before, would go on sending out inauthentic files in

the same way as before. Real users are flexible, and adapt themselves to different situations. If they see that many inauthentic files are moving on the network since informal norms regulating the P2P are not respected, it is likely that they would also start producing them, in order to gain credits, by an imitative behavior. While the use of reactive agents keeps the results more readable and easy comparable, in future works we'll implement cognitive ones, in order to explore their behavior under a RMS; they feature a policy which is dynamically created through trial and error, and progressive reinforcement learning. Two are the dimensions of learning that should be considered: one regarding the long term satisfaction of goals (related to the action of sending out an inauthentic file or not) and the other about the convenience in verifying a file (thus potentially losing a turn) related to the risk of being punished as an unwilling accomplice in spreading inauthentic files.

Besides, the threshold study now carried on at an aggregate level will be made also from the point of view of the individual agent: when does it become too costly to "cheat" for an agent so that it ceases to be beneficial? Such study will be made at a higher scale, referring to the number of agents and resources.

Also, if control through user collaboration has been studied, rewarding control should be considered as an individual incentive to control (with possible biases from malicious agent) and thus relate more to the collaboration objective of the study. This will also be studied in future works.

8. Acknowledgements

This work has been partially funded by the project *ICT4LAW*, financed by Regione Piemonte.

9. References

- [1] A. Josang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decis. Support Syst.*, 43(2):618–644, March 2007.
- [2] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 640–651, New York, NY, USA, 2003. ACM Press.

Exploiting the easyABMS methodology in the logistics domain

Alfredo Garro, Wilma Russo

Abstract— ABMS (Agent-Based Modeling and Simulation) has arisen as new approach to effectively support domain experts to cope with the growing complexity of the problems which they have to face and solve. To date, few methodologies are available which can be exploited by domain experts with limited programming expertise to model and subsequently analyze complex systems typical of their application domains. The easyABMS methodology has been proposed to overcome the lack of integrated methodologies able to seamlessly guide domain experts from the analysis of the system under consideration to its modeling and analysis of simulation results. In this paper, the effectiveness of easyABMS is demonstrated through a case study in the logistics domain which concerns the analysis of different policies for managing vehicles used for stacking and moving containers in a transshipment terminal.

Index Terms— Agent-Based Modeling and Simulation, Agent-Oriented Methodologies, Container Terminal Management.

I. INTRODUCTION

Agent Based Modeling and Simulation (ABMS) is a new approach for analyzing and modeling complex systems, an approach which is becoming acknowledged for its efficacy in several application domains (financial, economic, social, logistics, physical, chemical, engineering, etc) [17]. ABMS, allows for the definition of a system model based on autonomous, goal-driven and interacting entities (agents) organized into societies which is then simulated so to obtain significant information on not only the properties of the system under consideration but also its evolution.

Although several ABMS tools are currently available [10, 11, 19, 20, 24], there are only a few methodologies involving well- defined processes which are able to cover all the phases from the analysis of the system under consideration to its modeling and subsequent analysis of simulation results [7, 8, 17]. As a result, simulation models are often obtained using the two following approaches: (i) a direct implementation based on a chosen ABMS tool of the simulation model whose abstraction level is then too low and platform dependent as a conceptual modeling phase is not available; (ii) adapting a

given conceptual system model to a specific ABMS tool which, however, requires additional adaptation, calling for extra work, the amount of which increases depending on the gap between the conceptual and the implementation model of the system. Thus, both approaches lead to simulation models which are difficult to verify, modify and update.

To address these issues, a new methodology, easyABMS, has recently been proposed [4,5] which has specifically been conceived for agent-based modeling and simulation of complex system, seamlessly covering all the phases from the analysis of the system under consideration to its modeling and analysis of simulation results. easyABMS defines an *iterative* process which is *integrated*, *model-driven* and *visual*. In particular, each phase of the process refines the model of the system which has been produced in the preceding phase and its work-products are mainly constituted by *visual* diagrams based on the UML notation [23]. In addition, according to the *model-driven* paradigm [1, 21] the simulation code is automatically generated from the derived system Simulation Model. On the basis of the simulation results, a new/modified and/or refined model of the system can be obtained through a new process iteration which can involve all or some process phases.

Currently, easyABMS exploits the advanced features of visual modeling and of (semi)automatic code generation provided by the *Repast Symphony Toolkit* [17,18], a very popular and open source ABMS platform.

In this paper, the effectiveness of easyABMS in supporting domain experts to fully exploit the benefits of the ABMS, while significantly reducing programming and implementation efforts, is exemplified through a case study in the logistics domain. Specifically, the case study is focused on the analysis of different policies for the management of vehicles used for stacking and moving containers (*straddle carriers*) in a container transshipment terminal.

The remainder of this paper is organized as follows: Section II presents an overview of the easyABMS methodology and the related process; Section III presents a brief introduction to the reference application domain (a container transshipment terminal) and to related management problems; Section IV shows the application of easyABMS to the agent-based modeling and simulation of the Straddle Carrier Routing and Dispatching problem; finally, conclusions are drawn and future works delineated.

A. Garro is with the Department of Electronics, Informatics and Systems (DEIS), University of Calabria, Rende (CS), 87036 Italy. (e-mail: alfredo.garro@unical.it).

W. Russo is with the Department of Electronics, Informatics and Systems (DEIS), University of Calabria, Rende (CS), 87036 Italy. (e-mail: w.russo@unical.it).

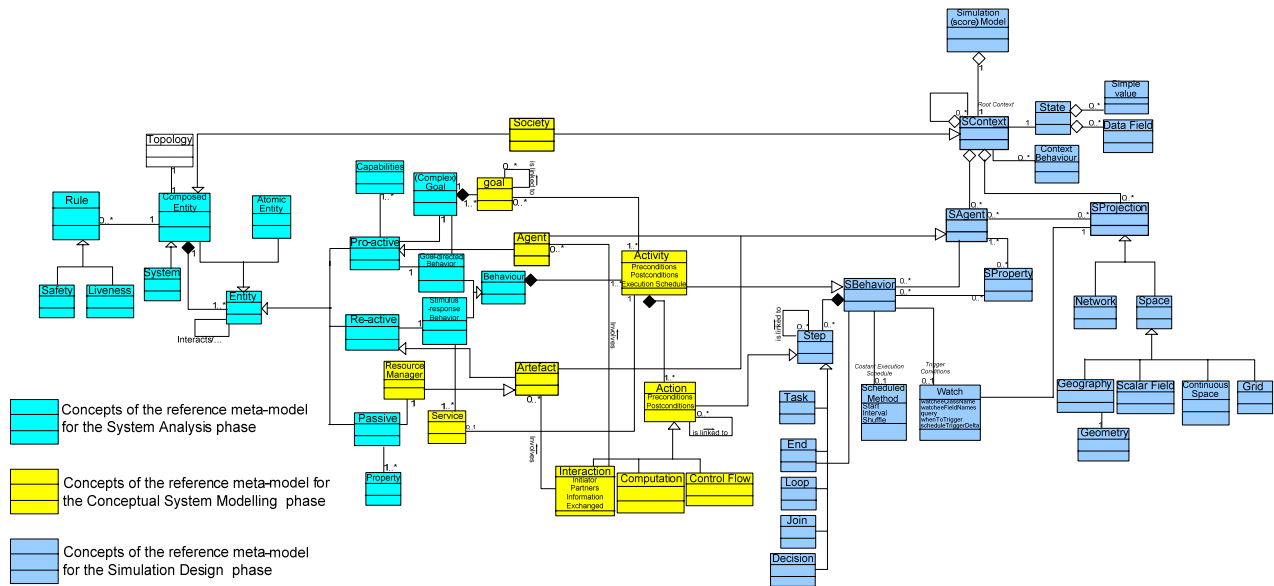


Fig. 1. The reference meta-model of easyABMS.

II. AN OVERVIEW OF EASYABMS

The easyABMS methodology defines an iterative process for ABMS composed of seven subsequent phases from the *System Analysis* to the *Simulation Result Analysis* [4, 5]. On the basis of the simulation results obtained a new iteration of the process which can involve all or some process phases can be executed for achieving new simulation objectives or those which have not yet been obtained. Specifically, the process phases are the following:

- *System Analysis*, in which a preliminary understanding of the system and the main simulation objectives are obtained (Analysis Statement);
- *Conceptual System Modeling*, in which a model of the system is defined in terms of agents, artifacts and societies (Conceptual System Model);
- *Simulation Design*, in which a model of the system is defined in terms of the abstractions offered by the framework which is exploited for the simulation (Simulation Model);
- *Simulation Code Generation*, in which the Simulation Code for the target simulation environment is automatically generated starting from the model which is obtained in the previous phase;
- *Simulation Set-up*, in which the Simulation Scenarios are established;
- *Simulation Execution and Results Analysis*, in which the simulation results are analyzed with reference to the objectives of the simulation previously identified in the *System Analysis* phase.

Currently, all the simulation related phases are supported by the *Repast Symphony Toolkit* [18, 20]. In particular, the *Simulation Design* and the *Simulation Code Generation* phases are supported by the *Repast Symphony Development Environment* [15], while the *Simulation Set-up*, the *Simulation Execution* and the *Simulation Results Analysis* phases are supported by the *Repast Symphony Runtime Environment* [16].

The models of the system generated by each process phase are produced according to the well-defined reference meta-model shown in Figure 1 so to facilitate the verification of the correctness of the models produced. Moreover, the concepts related to each phase are defined by extending and/or refining those of the previous phase; this allows for the seamless integration between the phases as the model produced in each phase extends and/or refines the model of the system produced in the previous phase.

The following sub-sections provides a brief description of each process phase.

A. System Analysis

In the *System Analysis* phase, the objectives of the simulation are specified and a preliminary understanding of the system and its organization is obtained.

This phase is based on the principle of *layering*, exploiting the well-known techniques of *Decomposition*, *Abstraction* and *Organization* [2, 9], and is constituted of a sequence of *analysis steps*. In each *step* a new system representation is produced by applying the *in-out zooming mechanisms* [12] to the entities comprising the system representation which resulted from the preceding *analysis step*. In the first *analysis step*, a starting level of abstraction for analyzing the system is chosen and then the system is *zoomed-in* so to identify its

component entities on the basis of the starting abstraction level.

According to the reference meta-model of the *System Analysis* phase (see Fig. 1), an *Entity* can be characterized by autonomous and goal-oriented behavior (*pro-active entity*), purely stimulus-response behavior (*re-active entity*), or can be *passive*. In addition, both the rules governing entities and their evolution, and the relationships among entities are specified. Specifically, *Safety* rules determine the *acceptable and representative* states of an entity whereas *liveness* rules determine which state transitions are feasible during entity evolution. Relationships can be either *intra-entity relationships* (i.e. relationships among the component entities obtained by the zooming-in of an entity) or *inter-entity relationships*.

The *System Analysis* phase ends when the user obtains a *System Representation* in which each component (pro-active, re-active, passive) entity has been represented at the level of abstraction which is appropriate for the objectives of the simulation. This *System Representation*, along with a synthetic description of the system being considered, a detailed description of each identified entity and the objectives of the simulation, constitutes the work-product of this phase (the *Analysis Statement*).

B. Conceptual System Modeling

In the *Conceptual System Modeling* phase, the *Structural System Model* is produced, and in particular, for each entity in the *System Representation*:

- the abstraction level suited to specific simulation objectives is chosen;
- the conceptual representation, in terms of *Agent*, *Artifact* or *Society*, is derived on the basis of the associations among the main concepts of the *System Analysis* and *Conceptual System Modeling* phases (see Fig. 1);
- the *interactions* with the other entities are obtained from the *intra* and *inter-relationships* where the latter cross the boundaries of societies.

The chosen level of abstraction of an entity can be modified in successive iterations through which it is then possible to produce new, modified, and/or refined *Structural System Models*.

For each entity in the produced *Structural System Model* a specific model is then defined, whose type can be one of the following depending on the entity type:

- **Society Model** which describes the entities which compose a *Society*, their type (*Agent*, *Artifact*, *Society*), and the rules governing the *Society* (*safety* rules) and its evolution (*liveness* rules);
- **Agent Model** which details the complex goal of an *Agent* (**Agent Goal Model**), its behavior as a set of *periodically scheduled and triggered Activities* (i.e. flow of *Actions*) which contribute to the achievement of the *Agent* goals (**Agent Behavioral Model**), and its interactions with other *Agents* and *Artifacts* in which the agent is involved (**Agent Interaction Model**);

- **Artifact Model** which describes the behavior of an *Artifact* as a set of *triggered Activities* related to the offered services (**Artifact Behavioral Model**), and its interactions with other *Artifacts* and *Agents* (**Artifact Interaction Model**).

C. Simulation Design

In this phase, starting from the *Conceptual System Model* a *Simulation Model* of the system, in terms of the abstractions offered by the framework exploited for the simulation, is produced.

In Figure 1 the basic simulation concepts of the reference simulation framework (the *Repast Symphony Toolkit* [18, 20]) are highlighted. Specifically, the central concept is the (simulation) *Context* (*SContext*) which represents an abstract environment in which (simulation) *Agents* (*SAgents*) can act and is provided with an internal *state* consisting of *simple values* and *Data Fields* (a n-dimensional field of values). In addition, an *SContext* can also support *behaviors* for the management of its internal *state*. *SContexts* can be organized hierarchically so to contain *sub-SContexts* which can have their own state. *SAgents* in an *SContext* can be organized by using *Projections* which are structure designed to define and enforce relationships among the *SAgents* in the *SContext*. In particular, a *Network Projection* defines the relationships of both acquaintance and influence between *SAgents* whereas *Space Projections* define (physical or logical) space structures (*Grid*, *Scalar Fields*, *Continuous Space*, *Geography*) in which the agents can be situated.

An *SAgent* can have multiple *behaviors* (*SBehaviors*), each operating on *SAgent Properties* and consists of a sequence of *Steps*; each *Step* can be associated with the execution of a *Task* or with the control of the flow of the *Task* execution (*Loop*, *Join*, *Decision*, *End*). Each *SBehavior* can be characterized by a *Scheduled Method* which defines a constant execution schedule, and by a *Watch* which periodically, on the basis of some *watched* parameters and conditions, triggers the execution of the behavior.

A *Repast Symphony* simulation model is defined by first specifying the structure and the characteristics of the *root SContext* and of all the possible nested *sub-SContexts*, in terms of their components (*SAgents*, *Projections* and *sub-SContexts*), and, then, specifying for each *SAgent* its *Properties* and *SBehaviors*, and for each *SBehavior* the component *Steps*, and the associated *Scheduled Method* and *Watch*.

The associations among the above described simulation concepts of the *Repast Symphony Toolkit* and the related concepts of the *Conceptual System Model* are reported in Figure 1. The exploitation of these associations makes it possible to directly obtain, starting from the *Conceptual System Model*, the *Simulation Model* of the System as follows:

- each *Society* becomes a *Repast Simulation Context* (*SContext*), the System is the root *SContext* and any enclosed *Society* is a (sub)-*Context* of the corresponding enclosing *Society*;

- *Artifacts* and *Agents* become Repast Simulation Agents (*SAgents*), the *Activities* which constitutes their behaviors are easily converted into Repast Simulation Behaviors (*SBehaviors*);
- *relationships* derived from *Interactions* among *Agents* and *Artifacts* generate *Repast Network Projections*.

D. The other Simulation related phases

According to the Model Driven paradigm [1, 21], the *Repast Symphony Development Environment* [15] is able to automatically generate a great part of the simulation code from the derived *Simulation Model* of the system. The simulation which can be extended with additional Java and XML code is then compiled by the *Repast Symphony Development Environment* using a Java compiler and then loaded into the *Repast Symphony Runtime Environment*.

The simulation executed by the *Repast Symphony Runtime Environment* can start after establishing: (i) the simulation scenario by specifying the values of the simulation parameters

defined in the *Simulation Design* phase; (ii) the presentation preferences for the simulation results concerning the system properties of interest identified during the *Simulation Design* phase.

Finally, the obtained simulation results can also be analyzed by exploiting the analysis tools (Matlab, R, VisAd, iReport, Jung) which can be directly invoked from the *Repast Symphony Runtime Environment* so to verify whether the objectives of the simulation identified during the *System Analysis* phase have been achieved. Where objectives have not been achieved or where new simulation objectives emerge, a new iteration of the process can be executed, which can then involve all or some process phases so that new/modified and/or refined models of the system can be produced for achieving the remaining/new simulation objectives.

TABLE I
MANAGEMENT PROBLEMS IN CONTAINER TRANSHIPMENT TERMINALS

PHASE	PROBLEM	DESCRIPTION
Arrival of the containership	QUAY CRANE ASSIGNMENT PROBLEM (QCAP)	Determining the number of quay cranes to assign to an incoming vessel.
	BERTH ALLOCATION PROBLEM (BAP);	Assigning incoming ships to berths, by taking into account constraints in both spatial and temporal dimensions so to minimize the time each ship spends in port (<i>turnaround time</i>).
Unloading and Loading of the ship	QUAY CRANE SCHEDULING PROBLEM (QCSP)	Determining a sequence of unloading and loading movements for cranes assigned to a vessel in order to minimize the vessel completion time as well as the crane idle time.
Transport of containers from the ship to the yard and vice versa	YARD MANAGEMENT	Allocating and reallocating the containers in the yard in order to reduce the amount of time required to handle of each vessel.
	STRADDLE CARRIER ROUTING AND DISPATCHING (SCRD)	Determining the operation to be performed by the straddle carries to maximize the productivity of each crane.

III. MANAGEMENT OF A CONTAINER TRANSHIPMENT TERMINAL

Due to the continuous growth in the volume of goods exchanged around world, further boosted by the rising Chinese and Indian economies, maritime transportation is becoming a crucial asset in global economy as it allows for large economies of scale in the transport sector. Specifically, the current maritime transportation system is based on a hub and spoke model [22] whereby ultra-large containerships operate between a limited number of mayor (mega)transshipment terminals (hubs), and smaller vessels (feeders) which link the hubs with other minor ports (spokes).

In this scenario, a hub terminal must maintain a high level of efficiency, not only to avoid traffic congestion but also to increase its competitiveness as some main characteristics (geographical, structural and technological) which also determine the competitiveness of a container terminal can be modified only on a long term perspective.

It thus becomes crucial to increase hub efficiency,

rendering it more competitive through the optimal management of terminal resources and optimizing tactical and operational logistics.

In the next sub-section, the organization of a maritime container terminal and some primary management issues are briefly discussed; a more complete description can be found in [13].

A. Organization of a Container Transshipment Terminal

Each ship approaching a maritime terminal enters in a harbour and waits to moor at an assigned berth position along the terminal quay which is equipped with giant cranes (quay cranes) for loading and unloading containers. These containers, in a *DTS (Direct Transfer System)* terminal, are transferred to and from the terminal yard by a fleet of vehicles (*straddle carrier*) which are able to stack containers in the yard. In contrast, in an *ITS (Indirect Transfer System)* terminal, containers are moved by trucks and trailers from the quay to the yard and vice-versa and staked by yard cranes.

In this context, the main logistic processes and related management problems can be grouped in relation to the flow

of containers in the terminal as shown and briefly described in Table 1; other issues are related to inter-terminal transportation and to possibly link with other transportation modes. Moreover, a transversal issue is related to the human resources management [13].

These very fundamental issues are not only reciprocally related, but the large-scale nature of hub management makes the use of standard exact solution algorithms impractical. In fact, the management of such large and intricately complex systems require new modeling methods which must also generate *proof-of-concept* simulations.

In the following Section, the effectiveness of the ABMS approach and the easyABMS methodology is shown focusing on the Straddle Carrier Routing and Dispatching Problem (SCRDP) [14]; with reference to the different management problems in a Container Transshipment Terminal (see Table 1), a more complete and domain specific agent-based simulator has been proposed in [6].

IV. MODELING AND SIMULATING STRADDLE CARRIER ROUTING AND DISPATCHING THROUGH EASYABMS

A. System Analysis

The main indicator of optimal performance in a container transshipment terminal is the *average ship-turn-around time* which is the average time-lapse between a ship's arrival and its departure, starting from the amount of time the ship waits for a berth (*berth waiting time*) and the duration for which the ship is docked for unloading and loading operations (*handling time*). In the following, the focus is set on the *handling time* given to fact that this time is highly dependent on the productivity of the Quay Cranes (QCs) and, as a consequence, on the management policies of the Straddle Carriers (SCs).

Specifically, to maximize the productivity of the QCs in a DTS container terminal, the SCs should operate so that the buffer of each crane, which has a limited capacity of only a few containers, is not full /not empty if the crane is performing the discharging/loading phase. Specifically, there are two main policies for organizing the work of SCs:

- *dedicated modality*: a given number of SCs are allocated to each QC to follow its working phases;
- *shared modality* (or *pooling*): a group of SCs is shared by two or more QCs which work on the same ship or on adjacent berthed ships and, possibly, frequently swapping between the tasks of loading and discharging containers.

The *shared modality* presents several benefits with respect to the *dedicated* mode: (i) reduction in the number of empty trips done by the SCs (i.e. travels without carrying any container), as the SCs can fruitfully alternate between trips carrying containers from the yard to the cranes which are loading outgoing cargo and trips back to the yard, carrying discharged cargo; (ii) more constant value of productivity of both QCs and SCs as, when a crane is not working, the SC of a pool can speed up operations of the other QCs.

A quantitative evaluation of the aforementioned benefits is not easy to obtain through traditional analytical models.

Moreover, classical dispatching models [14] often fail to provide dynamic assignment of container moves to SCs of a pool in order to speed up the loading/discharging operations (the Straddle Carriers Pooling Problem - SCPP). To overcome these shortcomings, an agent-based model can be defined and simulated with the following main objectives:

(i) quantifying the benefits of the *pooling* modality with reference to system productivity (vessels handling time) and cost reduction (numbers of exploited SCs and total distance covered);

(ii) obtaining an effective solution for the dynamic assignment of container moves to the SCs of a pool which can be used for automatically drive the coordinated behavior of the SCs in a real container terminal.

The *System Representation* obtained on the basis of the identified simulation objectives is reported in Figure 2. All the entities represented in Figure 2 are further described, along with their relationships and their *safety* and *liveness* rules, in a textual format enriched by tables and diagrams which are not reported due to space limitations.

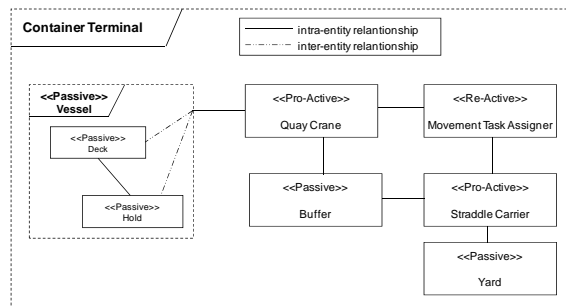


Fig. 2. System Representation

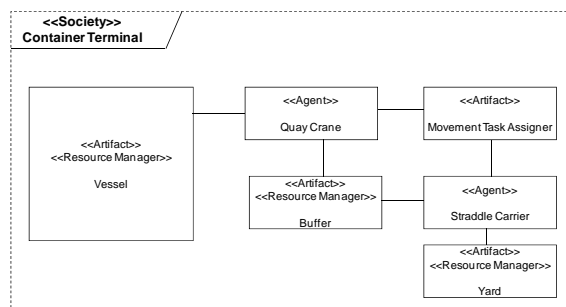


Fig. 3. Structural System Model

B. Conceptual System Modeling

The *Structural System Model* derived from the *System Representation* is reported in Figure 3; in particular, as the simulation objectives concern management policies of SCs, the level of representation chosen for the *Vessel* is more abstract with respect to the level resulting from the Analysis phase.

For each entity in the *Structural System Model* the corresponding *Society*, *Agent* or *Artifact Model* is defined (see Section II.B). Due to space limitations, the following subsections report only the *Society Model* for the *Container*

Terminal Society, the Agent Model for the Straddle Carrier Agent and the Artifact Model for the Movement Task Assigner Artifact.

1) The Container Terminal Society Model

The Society Model of the Container Terminal Society is shown in Figure 4 which reports the different entities which compose the Society, the safety and liveness rules which govern it and its dynamics.

Entity	Type
Vessel	Artifact (Resource Manager)
Quay Crane (QC)	Agent
Buffer	Artifact (Resource Manager)
Straddle Carrier (SC)	Agent
Movement Task Assigner	Artifact
Yard	Artifact (Resource Manager)

Safety rules

$$S_CTerm1. NCvi(t) = NCvi(t_0) - NCDvi(t) + NCLvi(t);$$

where $NCi(t)$ is the number of containers on the Vessel i at time t ; $NCDvi(t)$ is the number of containers that have been discharged from the Vessel i up to time t ; $NCLvi(t)$ is the number of containers that have been loaded onto the Vessel i up to time t .

$$S_Term2. \dots$$

Liveness rules

$L_CTerm1.$ A Quay Crane cannot download a container on its buffer if the buffer is full.

$$L_CTerm2. \dots$$

Fig. 4. The Society Model of the Container Terminal Society.

2) The Straddle Carrier Agent Model

Part of the Agent Model of the Straddle Carrier Agent is shown in Figure 5. In particular:

- Figure 5.a shows the Straddle Carrier Goal Model in which, as the two goals (Movement of containers from Buffer to Yard and Movement of containers from Yard to Buffer) can be achieved independently, no achievement relationship is present;
- Figures 5.b illustrates a part of the Straddle Carrier Behavioral Model; in particular, the Straddle Carrier Activity Table specifies the activities (Container Movement Activity) which the Straddle Carrier Agent executes for achieving its goals, along with the pre and post conditions and the execution schedule (periodical). Moreover, as the definition of an Agent Behavioral Model requires that each activity in the Agent Activity Table must be further described by an UML [23] Activity Diagram, the diagram for the Container Movement Activity is also shown. The UML Activity Diagram must be further enriched with an Activity Action Table (not shown in figure due to space limitations) which reports, for each single component action, a synthetic description of the action along with its pre and post conditions, the capabilities required for carrying out the action and its type (computation or interaction).
- Figure 5.c reports the Straddle Carrier Interaction Model which specifies, for each action of the interaction type (Task Assignment Request, Assigner Response) of the Container Movement Activity, the initiator, the partners of the interaction and the exchanged information.

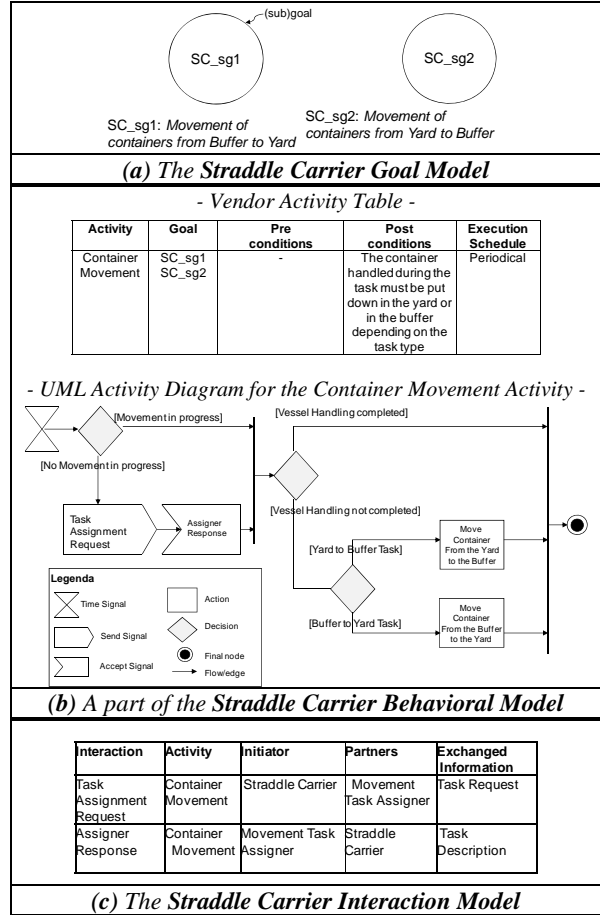


Fig. 5. Part of the Agent Model of the Straddle Carrier Agent.

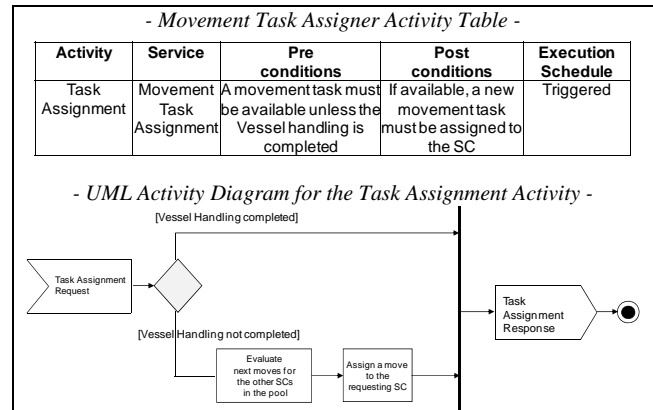


Fig. 6. Part of the Movement Task Assigner Behavioral Model.

3) The Movement Task Assigner Artifact Model

Figure 6 presents part of the Artifact Model of the Movement Task Assigner Artifact, and, in particular, the part of the Movement Task Assigner Behavioral Model which describes the Task Assignment Activity triggered by an SC requesting a new container movement to be performed. In particular, at the completion of its container movement the SC requests the next assignment from the Movement Task Assigner (see Figure 5.c). The Movement Task Assigner must

then decide, from available moves, the next best move for the requesting SC taking into account also subsequent moves which could be assigned to the other SCs in the pool (*Lookahead Policy*). Such planning could be dynamically revised at the next task assignment request.

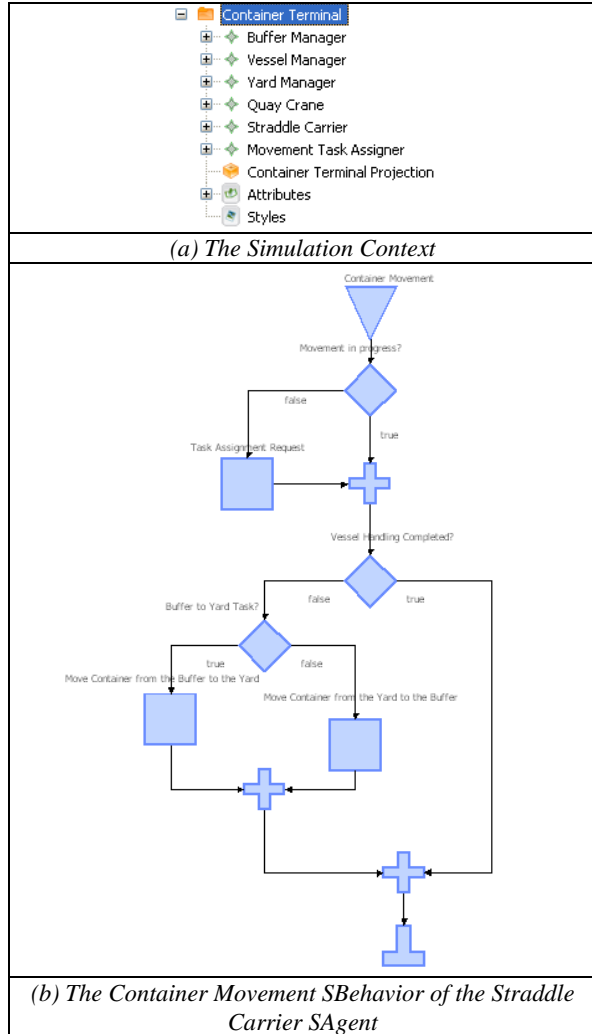


Fig. 7. Part of the Simulation Model.

C. Simulation Design

Figures 7.a-b show a portion of the *Simulation Model* produced by adopting the *Repast Symphony Toolkit* [18, 20] as the reference simulation framework. Figure 7.a shows the organization of the *Simulation Context (SContext)* whereas Figure 7.b shows a *Simulation Behavior (SBehavior)* of the *SAgent* representing a *Straddle Carrier*. In particular, the *Container Movement SBehavior* in figure 7.b corresponds to the *Container Movement Activity* reported in figure 5.b. The seamless transition between the two models is highlighted by the comparison between these two figures which clearly demonstrates that the behavior of an Agent/Artifact, defined during the *Conceptual Modeling* phase in terms of *Activities*

expressed by using the UML notation, can be directly mapped onto that of an *SAgent*, defined during the *Simulation Design* phase in terms of *SBehaviors*.

D. Simulation Execution and Results Analysis

Starting from the *Simulation Model* a great part of the simulation code is automatically generated by the *Repast Symphony Development Environment* [15], compiled by using a Java compiler and then loaded into the *Repast Symphony Runtime Environment* for the *Simulation Set-up* and *Execution*.

According to the simulation objectives, the execution of the resulting *Simulation Model* made it possible to compare and quantify the benefits of both *dedicated* and *pooling* modalities. In particular, several simulations have been executed for different scenarios in order to evaluate: the *Quay Crane Idle Time (QCIT)*, the *Straddle Carrier Covered Distance (SCCD)*, and the *Straddle Carrier Idle Time (SCIT)*. As an example, Figures 8.a-b illustrate the *QCIT* and the *SCCD*, in the two different modalities, with reference to a simulation scenario based on real-life organizational topology and equipment typologies of the *Gioia Tauro Container Terminal* [3]. In this simulation scenario one *Vessel* is handled by two *QCs* for the loading and discharging of 50 containers respectively. The results shown in Figure 8, which are results averaged from 30 simulation runs, made it possible to quantify the significant advantage of the *pooling* modality in terms of vessel handling time and cost reduction.

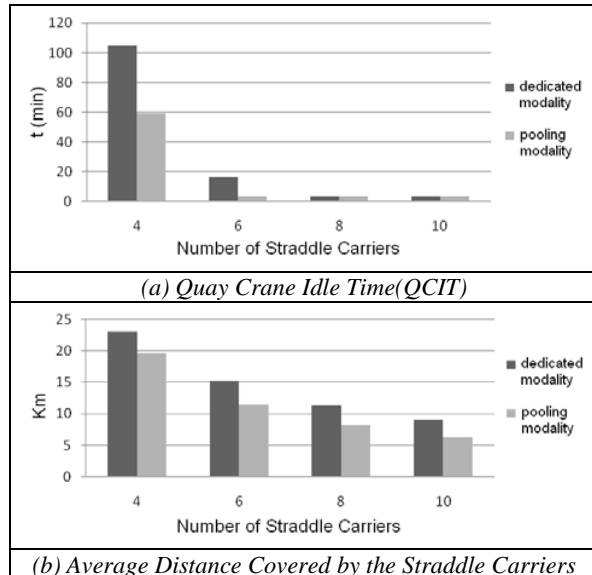


Fig. 8. Some Simulation Results.

V. CONCLUSION

Several tools for ABMS are now available as well as methodologies for the development of agent-based systems which are mainly proposed in the context of *Agent-Oriented Software Engineering (AOSE)*. Nonetheless, only a few results are available which integrate the methodological

features coming from the AOSE with the modeling and simulation features of modern ABMS tools. As a consequence, scarce support in the whole process which goes from the system analysis to the analysis of simulation results is provided to domain experts with limited programming expertise. To address these issues, easyABMS, a recently proposed and full-fledged methodology for agent-based modeling and simulation of complex systems, fruitfully exploits both AOSE modeling techniques and simulation tools specifically conceived for ABMS.

In this paper, the effectiveness of easyABMS has been demonstrated using a case study in the logistics domain which concerns the analysis of different policies for managing *Straddle Carriers* in a *Container Transshipment Terminal*. In particular, overcoming the main limitations when using only classical analytical models, a quantitative assessment of two primary *Straddle Carrier* management policies and an effective solution in guiding the dynamic assignment of container moves have been easily provided. The exploitation of easyABMS allowed to demonstrate how this new methodology can seamlessly guide domain experts from the analysis of the system under consideration to its modeling and simulation, as the phases which compose the easyABMS process, the work-products of each phase, and the (seamless) transitions among the phases are fully specified. In addition, easyABMS focuses on system modeling and simulation analysis rather than details related to programming and implementation as it exploits the Model Driven paradigm, making it possible the automatic code generation from a set of (visual) models of the system.

Future research efforts will be devoted to: (i) extend the Repast Symphony Toolkit so to obtain an integrated ABMS environment which fully supports all the process phases also comprising the System Analysis and Conceptual System Modeling phases; (ii) extensively experiment easyABMS in case studies of social, financial, economic, and logistic relevance; (iii) adopting a meta-simulation framework for the Simulation Design phase so to obtain a platform-independent simulation model which can then be translated into different platform-dependent simulation models.

REFERENCES

- [1] C. Atkinson and T. Kühne. Model-driven development: A metamodeling foundation. *IEEE Software*, 20(5):36-41, 2003.
- [2] G. Booch. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley, 1994.
- [3] J.-F. Cordeau, M. Gaudioso, G. Laporte, and L. Moccia. The service allocation problem at the Gioia Tauro maritime terminal. *European Journal of Operational Research*, 176:1167-1184, 2007.
- [4] A. Garro, W. Russo. An integrated agent-based process for the simulation of complex systems. *Proceedings of the International Conference on Economic Science with Heterogeneous Interacting Agents (ESHIA)*, Warsaw, Poland, 19-21 June, 2008.
- [5] A. Garro, W. Russo. An integrated and iterative process for agent-based modeling and simulation. *Proceedings of the 5th International Conference of the European Social Simulation Association (ESSA)*, Brescia, Italy, 1-5 September, 2008.
- [6] L. Henesey, P. Davidsson, and J. A. Persson. An agent-based approach for building complex software systems. *Autonomous Agents and Multi-Agent Systems*, 18(2):220-238, 2009.
- [7] T. Iba and N. Aoyama. Understanding Social Complex Systems with PlatBox Simulator. In *Proc. of the 5th International Conference on Computational Intelligence in Economics and Finance (CIEF2006)*, pages 64-67, Taiwan, October 2006.
- [8] T. Iba, Y. Matsuzawa, and N. Aoyama. From Conceptual Models to Simulation Driven Development of Agent-Based Simulations. In *Proc. of the 9th Workshop on Economics and Heterogeneous Interacting Agents*. Kyoto, Japan, 2004.
- [9] N. R. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35-41, 2001.
- [10] MASON Home Page. George Mason University, Fairfax, VA, available at <http://cs.gmu.edu/~eclab/projects/mason/>.
- [11] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations. Working Paper 96-06-042. Santa Fe Institute, 1996.
- [12] A. Molesini, A. Omicini, A. Ricci, E. Denti. Zooming multi-agent systems. In *Proc of the 6th International Workshop on Agent-Oriented Software Engineering (AOSE 2005), AAMAS 2005*, Utrecht, The Netherlands, 2005.
- [13] M. F. Monaco, L. Moccia, and M. Sammarra. Operations Research for the management of a transshipment container terminal. The Gioia Tauro case. *Maritime Economics and Logistics*, 2009, Vol. 11, n. 1, pp. 7-35.
- [14] M. F. Monaco, and M. Sammarra. Scheduling and dispatching models for routing straddle carriers at a container terminal. *Proceedings of the XXXVIII Annual Conference of the Italian Operations Research Society*, Genova, Italy, Sept. 5-8, 2007.
- [15] M. J. North, T.R. Howe, N.T. Collier, and J.R. Vos. The Repast Symphony Development Environment. In *Proc. of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms*, Chicago, IL, October 2005.
- [16] M. J. North, T.R. Howe, N.T. Collier, and J.R. Vos. Repast Symphony Runtime System. In *Proc. of the Agent 2005 Conference on Generative Social Processes, Models, and Mechanisms*, Chicago, IL, October 2005.
- [17] M. J. North, C. M. Macal. *Managing Business Complexity: Discovering Strategic Solutions with Agent-Based Modeling and Simulation*. Oxford University Press, 2007.
- [18] M.J. North, E. Tatara, N.T. Collier, and J. Ozik. Visual Agent-Based Model Development with Repast Symphony. In *Proc. of the Agent 2007 Conference on Complex Interaction and Social Emergence*. Northwestern University, Evanston, IL, November 2007.
- [19] M. Resnick. *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Multi-agent Systems*. MIT Press, Cambridge, Mass., 1997.
- [20] ROAD (Repast Organization for Architecture and Design). Repast Home Page, Chicago, IL, available as <http://repast.sourceforge.net/>.
- [21] D. C. Schmidt. Model-Driven Engineering. *IEEE Computer*, 39(2):41-47, 2006.
- [22] UNCTAD (2006). *United Nations Conference on Trade and Development - Review of Maritime Transportation*.
- [23] Unified Modeling Language (UML) Specification. Version 2.1.2. Object Management Group Inc., 2007.
- [24] U. Wilensky. NetLogo. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 1999.

Engineering Development of Agents using the Cooperative Behaviour of their Components

Noélie Bonjean, Carole Bernon, Pierre Glize
 Institut de Recherche en Informatique de Toulouse
 University of Toulouse III
 118 route de Narbonne, 31062 Toulouse cedex 9, France
 Email: Firstname.Name@irit.fr

Abstract—The objective of the work undertaken here is to endow an agent-oriented methodology (such as ADELFE) with a semi-automatic tool for helping designers when drawing up the agents composing an adaptive multi-agent system (AMAS). This tool acts as a guide for enabling designers to influence the emergent global behaviour of an AMAS by acting on the local behaviour of its cooperative agents. The preliminary approach proposed in this article can be seen as a feasibility study aiming at developing a textual guide by considering the principles of the AMAS theory. Simulation of the behaviour of healthy and cancerous cells is used as a base for this study.

I. INTRODUCTION

Multi-Agent Systems (MAS) are a recognised paradigm for designing and implementing complex applications and several agent-oriented methodologies were proposed to guide engineers in such a task [1] [2]. However when complexity prevents designers from discovering an *a priori* algorithm, designing MAS may also be a complex problem and new approaches may be adopted, such as bottom-up ones. Agents composing a MAS are identified, their behaviour and interactions defined to let them build the actual organisation of this MAS and let the global collective behaviour emerge from these interactions. This approach is adopted when building Adaptive Multi-Agent Systems (AMAS) in which every agent has a cooperative attitude [3]. However this emergence at the macro-level does not prevent engineers from having difficulties for finding the right micro-level cooperative behaviours and helping them is still an issue. ADELFE¹ was proposed as a guide dedicated for designing AMAS, but it has still some lacks. Therefore, additional guidelines and tools have to be provided for enriching it.

The objective of the work undertaken here is then to endow ADELFE with a semi-automatic tool for helping engineers when drawing up the agents composing an AMAS. This tool acts as a guide for enabling designers to influence the emerging global behaviour of an AMAS by acting on the local behaviour of its cooperative agents. The approach proposed in this article is still a preliminary one. It can be seen as a feasibility study aiming at developing a textual guide to facilitate designing agents. This textual guide is developed by studying an application related to the biological domain and simulating the behaviour of healthy and cancerous cells.

Section II presents the context for this study and Section III positions it according to existing works. The simulation adopted as a base for the proposed approach is presented in Section IV. This enables expounding how developing such a simulation may be guided in Section V before concluding with some prospects.

II. CONTEXT

The aim of this study is to facilitate the design of agents composing a specific type of MAS, adaptive ones. This section introduces the concepts to which this study is related.

A. AMAS Theory and Cooperative Agents

Adaptive multi-agent systems at the heart of this study apply self-organisation principles according to the AMAS theory detailed in [3]. According to it, for designing a system whose functionality is adequate with what is expected by the designer, it is sufficient to drawing this system up with parts which have a cooperative attitude. This cooperative attitude means always trying to avoid, otherwise repair, situations that are judged, from the own point of view of an agent, as non cooperative. An agent does this by changing its relationships with other agents. This also changes the internal organisation of the multi-agent system it belongs to and as a result transforms the collective function the system is performing, making it adaptive.

A behavioural model of a cooperative agent was proposed and used during the microMega² project which aim was to model and simulate the behaviour of a unicellular microorganism [4]. Adopting such a model has firstly simplified the visualisation of the different parts composing the behaviour of a cooperative agent (see Fig. 1). Indeed, this model separates the nominal behaviour of a cooperative agent from its adaptive (or cooperative) one, this latter being itself broken into tuning, reorganisation and evolution behaviours. A designer may then work on each part of the agent's behaviour almost independently and implement and test them in a gradual way. These behaviours may be described as follows:

- the nominal behaviour represents the basic behaviour of an agent, what it does for achieving its local function without necessarily coping with Non Cooperative Situations (NCS),

¹Atelier de Développement de Logiciels à Fonctionnalité Emergente

²National ANR-funded project, 2005-2008

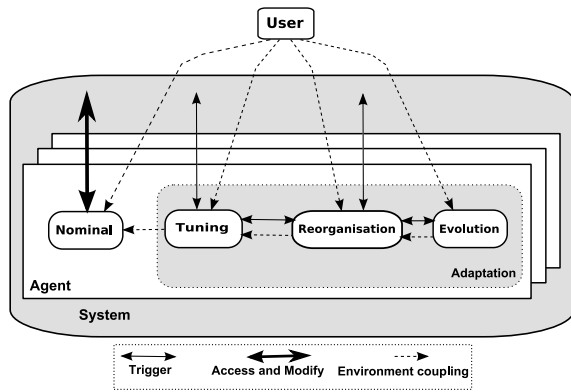


Fig. 1: Behavioural model of a cooperative agent.

- the adaptive behaviour, added on top of the nominal one, aims at dealing with these cooperative failures in three different ways:
 - by trying to adjust the values of the parameters used during the nominal behaviour (tuning behaviour),
 - by changing its relationships with others for trying to solve dead-ends (reorganisation behaviour),
 - and finally by self-removing or creating other agents if NCS still remain (evolution behaviour).

B. ADELFE and Living Design

For assisting engineers when designing AMAS, ADELFE was proposed. During the design phase, a specific activity is provided as a guide for designing agents: every non cooperative situation has to be identified and every preventing or repair action has also to be defined. This textual guide is still insufficient because currently, nothing in ADELFE guarantees that this identification is the proper and complete one. It is therefore necessary to enrich ADELFE with a better tool.

Ideally this automatic (or semi-automatic) tool would enable to develop the adequate behaviour for a cooperative agent, the very one that would allow achieving the functional adequacy of the AMAS this agent belongs to. This tool should also take into account the benefits brought by the behavioural model presented in the previous section by enabling designers to separately act on the different parts composing such a behaviour.

The following section presents this issue and gives insight into existing works that have links with this one.

III. THE PROBLEM

Considering the model given in Fig. 1, each part of the behaviour of an agent has an action on the other one. A NCS occurring during the nominal behaviour may trigger a repair action that will be performed during the adaptive behaviour. Depending on the designer's degree of familiarity with the AMAS concepts, the behaviour of an agent may be devised in two ways:

- The designer succeeds in identifying non cooperative situations. He certainly designs the nominal behaviour

of agents by including some actions that enable an agent to prevent or repair non cooperative situations it will encounter. Furthermore, he is also capable of separating the nominal behaviour from the adaptive one and implements an agent according to the behavioural model presented above.

- In the second case, this designer does not really know how to identify non cooperative situations for the agent he is designing, or this identification is incomplete. Furthermore, he does not know how to define the actions this agent has to perform for staying in a cooperative state. The nominal behaviour he designs essentially consists in the basic behaviour of this agent and does not concern the aspects related to its cooperative attitude. Such a designer encounters difficulties for finding the behavioural dichotomy the model lays down.

In both cases finding every non cooperative situation an agent may encounter is not guaranteed, let alone defining every action required for removing such situations. Theoretically this would result in the inability of agents to collectively achieve a functionally adequate global function: what emerges from their interactions will not suit the designer's expectations. Ideally ADELFE should help designers for verifying these points and/or complete the agent design they made.

A. Simulating for Designing?

In 2002, using simulation for AOSE and designing agents was a challenge as underlined in [5]. For a few years now, steps are made in this direction with several research works using simulation in the AOSE domain [6]–[8].

Some works were made in this sense for enriching ADELFE as mentioned above. Simulation has been used for automatically detecting NCS in an AMAS prototype [9]. Simulation has also be used for making agents self-adjust their behaviour by making their behavioural rules self-reorganise [10]. Although both approaches had conclusive results for rather simple applications, they nevertheless have some drawbacks. The kind of agents that was taken into account (which are situated in an environment and communicate in an indirect manner through this environment) and the underlying use of the SeSAM platform [11] (need to become familiar with it, performance problems when a great number of rather complex agents are simulated) may be considered as limitations. Furthermore enabling a designer to make a prototype of his system before really implementing it is interesting; however, our main aim is rather to make a first step towards enriching ADELFE with a “good practice guide” before (semi-)automating this guide through an appropriate tool.

B. The Adopted Approach

The issue here is therefore to study how general principles may be extracted from the features of AMAS, and more especially those of cooperative agents, in order to deliver this guide.

An AMAS simulating a behaviour is used as the base for this study. A designer of MAS, who is considered as being

unfamiliar with the way of implementing AMAS, follows ADELFE and designs agents for achieving a first functional version of this system. This version, probably only based on the nominal part of agents, will likely not be the proper one. Therefore, the second step is to try and improve this implementation by acting on the different parts of the adaptive behaviour an agent may have.

IV. CASE STUDY: SIMULATING CELLS

To stay in line with the application domain of microMega, and because biological phenomena are complex and dynamic, a simulation inspired from the biological world was chosen. Actually, simulation and modelling in biology are a very active field of research, notably when cells are involved [12]. Mathematical models, based on differential equations e.g., may be used [13] or less classical tools such as Petri nets [14], cellular automata [15] or neural networks. For a few years now, MAS are also contributing to this effort, mainly because they are able to scale and to model specific properties in a more comprehensible way than mathematical models [16]. Among these works, some are more precisely interested in simulating the behaviour of microorganisms or cells [17]–[19].

This section therefore describes the features of the target application and how it was implemented.

A. Description of the Simulation

In a living organism, mitosis enables division of a eukaryotic cell into two daughter cells and apoptosis makes such a healthy cell degrade and die. Mutations may also occur (due to radiations, viruses, genetic predispositions, etc.) by copying errors of genetic material during division. Most of the time, mutations are not harmful for the cell or are repaired by internal mechanisms. However, some changes may lead to malignant cells leading to cancer. Among other properties, a cancerous cell is not able to repair DNA anymore, divides in an unchecked way and does not die. Cells “communicate” with their environment by endocytosis, which enables them to absorb material from outside, or exocytosis, by secreting material to the extracellular environment. Molecules which may be released are moving through the interstitial tissue. Some of them play a role in the paracrine signalling by carrying information from a cell to another one [20]. Some molecules also supply cells with energy. In both cases, moving make them degrade with time.

According to this simplified biological context, the aim of the considered application is to simulate the behaviour of healthy and cancerous cells. The development process of ADELFE was applied for analysing the requirements, verifying the AMAS adequacy and identifying the agents involved in this MAS, and designing their nominal behaviour with respect to the basic biological knowledge previously presented. Due to lack of space and because the objective of the paper is not to focus on it, only some details are given.

Because cells are autonomous, have a local goal (survive for healthy ones and proliferate for malignant ones), have only a partial view of their environment, two types of agents were

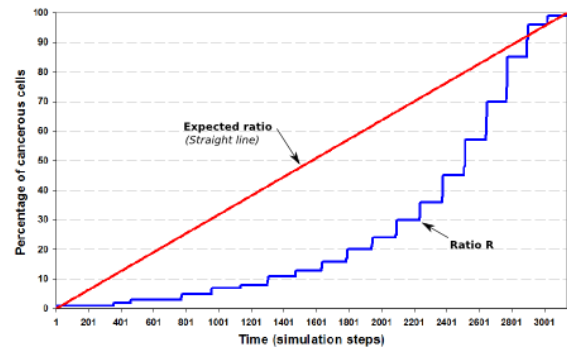


Fig. 2: Ratio of cancerous cells over time.

identified : healthy cells and cancerous ones. Molecules are moving in an autonomous way, have also a limited perception of their environment but they do not have a local goal and are not considered as agents. Message passing simulates the absorption of a paracrine molecule by a cell and the related communication. Healthy cells may divide, mutate, die, signal molecules, or detect and absorb molecules. Cancerous cells do not mutate and die.

B. Collective Nominal Behaviour Obtained

The designer of this application is regarded as being able to deal with only the nominal behaviour of agents. As a result, the nominal behaviour of cells was implemented and the adaptive behaviour is what the guide has to help finding.

To see how the collective behaviour obtained for this MAS fits a user’s needs, the ratio R of cancerous cells was studied. $R = \frac{C}{C+H}$ where C is the number of cancerous cells and H , the number of healthy ones. The curve obtained for R is shown on the right part of Fig. 2.

A healthy cell has a given probability of mutation and proliferates only if some free space is near it. On the contrary, a cancerous cell tries to push healthy ones if it does not have available space around it and divides itself if it succeeds. As a result, a slow increase of the number of cancerous cells can be seen on the curve at the beginning of the simulation because few mutations occur yet. However, once several malignant cells appear, they proliferate in a faster way than healthy ones, also because they do not die, and invade the tissue with time.

Let us suppose now that this collective behaviour is not the one expected by the end-user of this simulated system, e.g., a biologist. This user would like to obtain a curve with another shape; for instance, one that would express a more regular appearance of cancerous cells which is shown on the left part of Fig. 2. This shape has no biological reality and was chosen only for studying how the engineer could be guided to adapt the local behaviour of cells in order to obtain the new expected collective behaviour.

V. GUIDING THE SIMULATION

Actually, changing the behaviour of an agent may be done by changing the basic function it performs, its nominal behaviour. However, this choice is contrary to the hypothesis

we previously made; only the adaptive behaviour has to be built or modified. This means modifying one or all the parts of this behaviour: tuning, reorganisation and evolution. Since this study is a first step towards our goal, only the tuning part will be reviewed.

A. Act on the Tuning Behaviour of Agents

In general, the tuning behaviour of an agent has to be modified in order to get the expected global behaviour (the expected shape for the studied curve, e.g.). Modifying the value of the parameters that are used by an agent is going to modify how the nominal behaviour, which uses these very parameters, performs and therefore is going to influence the global behaviour of its collective. The issue lies in identifying the right parameters that have to be calibrated in order to get the proper behaviour.

The approach proposed is to study first how the parameters of an agent may promote its possible actions. Which actions have to be influenced (fostered or encouraged) in order to get the expected behaviour is then determined. Since a parameter may influence one or several actions, these relationships have to be gradually propagated for determining the proper parameter to change in order to modify the actions previously identified.

B. Relate Parameters and Actions of Agents

By leaning on the application domain knowledge, the designer has to establish a table relating the actions agents may perform and the parameters these agents use. Actually, this table should be done before beginning implementation, it is also a means to see how agents behave and to reflect on what they perform according to what they have to do.

Such tables were built for healthy cell agents (see Table I) and cancerous ones (see Table II). In this application, 18 parameters may influence the 7 actions of healthy cells:

- A1 - Proliferate,
- A2 - Mutate,
- A3 - Die,
- A4 - Absorb a paracrine molecule,
- A5 - Signal a paracrine molecule,
- A6 - Absorb a “simple” molecule,
- A7 - Signal a “simple” molecule.

or the 5 ones of cancerous cells (A1', A4', A5', A6' and A7'). Due to lack of space, out of the 18 parameters, 12 parameters are considered in the tables built, those that are only required for the coming reasoning:

- P1 - Lifetime of a cell: lifetime in the means case,
- P2 - Energy of a cell: internal energy a cell has for living,
- P3 - Molecule concentration in the environment: number of molecules a cell may perceive,
- P4 - Environmental conditions: occupied spaces surrounding a healthy cell and free ones for a cancerous one,
- P5 - Proliferation speed: expressed in simulation steps,
- P6 - Energy cost for proliferating: energy required for this action,

TABLE I: Influence table for healthy cell-agents.

Param/Action	A1	A2	A3	A4	A5	A6	A7
P1	+		--				
P2	+++	++			++	++	-
P3	+			+++	--	--	+++
P4	--	+++	+++				
P5	++		-				
P6	-		++				
P7			+		-		
P8			+				-
P9			+			-	
P10			++				
P11		+		+++			
P12		+++					
...							

- P7 - Energy cost for signalling a paracrine molecule,
- P8 - Energy cost for absorbing a molecule of any type,
- P9 - Energy cost for signalling a “simple” molecule,
- P10 - Apoptosis signals: signals which tend to influence death when overpopulation occurs,
- P11 - Sensitivity threshold to paracrine: expresses the ability of a cell to react to paracrine molecules,
- P12 - Mutation rate: number of cells that may mutate during a lifecycle.

For each (parameter, action) pair, two criteria are represented in these influence tables:

- In which direction (increase/decrease) the given parameter may vary for promoting the action. This is expressed by + or –.
- The influence of the given parameter on the action. This is expressed by the number of symbols + or – used.

For example, the parameter *P1-Energy of a cell* has to increase for promoting the action *A1-Proliferate* of a healthy cell and this influence on *A1* is high because three + are used. *P1* has also an impact on the action *A7-Signal a “simple” molecule*. In order to encourage a healthy cell to make this action, *P1* has to be decreased and this influence is low because only one – is used.

Once these tables designed, a reasoning has to be done in order to modify the behaviour of cell-agents.

C. Propagate Influences

A parameter may influence several actions (*P1* influencing *A1* and *A7*, e.g.), and by modifying it, the related actions will also be promoted or discouraged, depending on the impact of this parameter. Therefore before modifying a parameter, studying how its influences are propagated from an action to another one is required. The influence tables previously built will be a help for this study.

For trying to obtain a more linear curve for *R*, starting from Fig. 2, *R* has to increase when *R* is below the straight line which represents its expected value and it has to decrease when it is above this line.

TABLE II: Influence table for cancerous cell-agents.

Parameter/Action	A1'	A4'	A5'	A6'	A7'
P1					
P2	+++		++	++	-
P3	+	+++	--	--	+++
P4	++				
P5	++				
P6	-				
P7			-		
P8					-
P9				-	
P11		+++			
...					

In a first step, R is going to be increased. Since R is a fraction, it increases when its numerator does and/or its denominator decreases. Since C cannot decrease, C has to increase or S has to decrease. To make C increase and S decrease, the action $A1$ -Proliferate has to be promoted for cancerous cells, $A1'$ -Proliferate has to be repressed for healthy ones and $A2$ -Mutate as well as $A3$ -Die have to be promoted for healthy cells.

Each action has to be analysed depending on the relationships it has with the parameters and how this influence propagates.

1) *Promote "A1'-Proliferate" for Cancerous Cells:* Proliferation of cancerous or healthy cells are promoted by the same parameters except $P4$ -Environmental conditions. Therefore, promoting $A1'$ by playing on $P1$, $P2$, $P3$, $P5$ or $P6$ will also promote $A1$, and $A1$ has to be deserved. These parameters cannot be modified. The $P4$ parameter represents the available space around a cell and is calculated by $(NCC + NHC + AS) - (NCC + NHC)$ where NCC is the number of neighbouring cancerous cells, NHC is the number of neighbouring healthy cells and AS is the number of available spaces around the cell. This formula should give the highest possible result for promoting $A1'$.

2) *Promote "A1-Proliferate" for Healthy Cells:* As above, some parameters cannot be changed. However other parameters have an impact on $A1$ and do not influence $A1'$:

- $P1$ -Lifetime of a cell has a low influence on $A1$. Deserving it should be obtained by decreasing $P1$ and this would strongly promote $A3$ -Die which is the expected effect.
- $P4$ -Environmental conditions has a high influence on $A1$. It represents the occupied spaces around an healthy cell. $P4$ is increased by promoting $A1/A1'$ for each kind of cells or by deserving $A3$. The former alternative is contrary to the expected effect for healthy cells and redundant for what was done above for cancerous ones. The latter alternative is the opposite of the expected effect. Another solution would be to compute the value of $P4$ in a different way. The current formula is $(NCC + NHC)/(NHC + NCC + AS)$. By adding the respective coefficients a , b and c to NCC , NHC and AS in order

to reflect their importance, a new formula may be used: $(a \times NCC + b \times NHC)/(a \times NCC + b \times NHC + c \times AS)$. Furthermore, increasing this parameter would also promote, with the same importance, the actions $A2$ -Mutate and $A3$ -Die and this goes in the right direction.

3) *Promote "A3-Die" for Healthy Cells:* This action is influenced by $P1$ -Lifetime of a cell and $P4$ -Environmental conditions which were studied before. Some other parameters are also involved (from $P5$ to $P10$). $P10$ -Apoptosis signals influence solely this action and acting on them may be easy. Energy costs (from $P6$ to $P9$) could be increased for promoting $A3$ -Die. However, by studying propagation, these costs have also an impact on other actions such as absorption and signalling of molecules, and this could modify $A2$ -Mutate; on the other hand, $A1$ -Proliferate could also be deserved, and so on. Not modifying these parameters seems to be a good choice to try and avoid a too great influence on the system.

4) *Promote "A2-Mutate" for Healthy Cells:* Among the parameters that were not examined before, $P12$ -Mutation rate has no influence on any other actions than $A2$, therefore it may be modified as needed. Adjusting $P11$ -Sensitivity threshold would be propagated on $A4$ -Absorb paracrine without any harmful effect on the system.

Once this propagation studied, some parameters may vary while others cannot. According to these new conditions, modification of some of the potential modifiable parameters has then to be done.

D. Tune Parameters

Table III shows the parameters that may vary (in bold font) or not (normal font) for *promoting* (+ before the name of an action) or *deserving* (-) actions of healthy or cancerous cells (their specific parameters or actions are distinguished from those of healthy cells by a ' following their name).

Parameters for which propagation is not opposite to the expected result are first modified. $P1$, $P4$, $P11$ and $P12$ are chosen, especially because they have the highest influence. The formula related to $P4$ is thus modified for making cancerous cells have a higher impact and becomes $(2 \times NCC + NHC)/(2 \times NHC + NCC + AS)$.

E. Verify the Impact of Tuning

Once the above reasoning done and parameters modified in the simulation code, a new kind of curve for R was obtained (see Fig. 3). The first part of this curve is more linear than previously and gets closer to a straight line. However, its second part (from step 800) does not properly fit expectations.

Therefore, the same reasoning has to be done in order to make the top curve as linear as possible by decreasing R without cancelling the work done previously. An opposite operation has to be done: deserve $A1'$ -Proliferate for cancerous cells, $A2$ -Mutate and $A3$ -Die for healthy ones, and promote $A1$ -Proliferate for healthy cells. Since the benefits brought by the previous adjustment have to remain, the parameters that may be modified are those that were still untouched. In this case, only $P4'$ was not modified and the value of environmental conditions for cancerous cells is then turned down.

TABLE III: Influence table for making R increase.

Parameter/Action	- A1	+ A2	+ A3	+ A1'
P1	-		- - -	
P2	- - -	++		+++
P3	-			+
P4	+++	+++	+++	
P4'				++
P5	- -		-	++
P6	+		+	-
P7			+	
P8			+	
P9			+	
P10			++	
P11	+			
P12		+++		

F. Observe the Collective Behaviour Obtained

After these two adjustments, a typical final curve obtained is shown in Fig. 4. It is more linear and looks like the one end-users expected. Therefore the collective behaviour of the cells is more in line with the expected one. As expected, the shape of the curve shows that cell-agents have modified their behaviour in the right direction.

It is also worth observing that the collective behaviour before and after modifications made on the tuning behaviour of cells does not give the same spatial distribution for cancerous cells, as shown in Fig. 5. In this figure, healthy cells are empty hexagons, cancerous ones are black hexagons and paracrine molecules are dots between cells. To simulate an “infinite” tissue, cells on opposite borders are neighbours. These snapshots were taken when around 10 cancerous cells appeared. They show that cancerous cells tend to be more scattered in the simulated tissue when only the nominal behaviour was concerned. After tuning, they tend to form clusters because these cancerous cells can mainly be created at the frontiers of clusters, and this prevents their exponential development. Of course as time goes by, the number of cancerous cells increases and their distribution tends to be more dense in both cases. However the initial disparity is consistent with the results and the aim of obtaining a straight curve.

Furthermore, this disparity and clusterisation was obtained without reasoning on this macro-level; only by studying relationships between parameters and actions *inside* the agents. Here also, there is emergence of a phenomenon at the macro-level which was controlled by cooperation at the micro-level. More precisely, there are three levels:

- 1) The macro-level which corresponds to the cell tissue. It is used to observe the global behaviour and to enter feedbacks according to the end-user wishes.
- 2) The meso-level constituted by the individual cells, where *ad hoc* feedbacks arrive according to their type (cancerous or not).
- 3) The micro-level corresponding to the cell components, where cooperative “negotiations” (as the “good practice

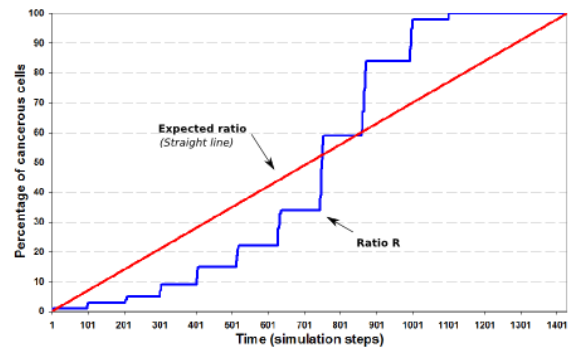


Fig. 3: New ratio obtained after the first tuning.

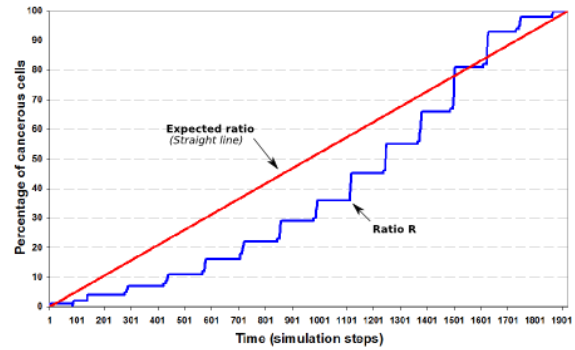


Fig. 4: Final ratio obtained.

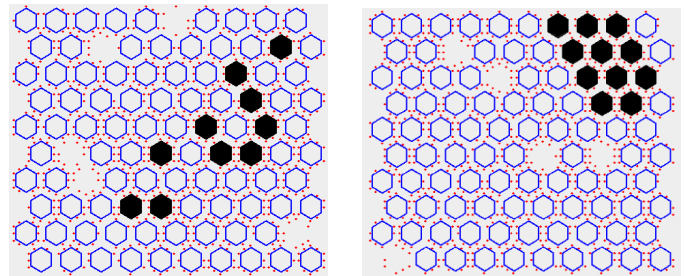


Fig. 5: Example of distribution obtained before (left) and after (right) tuning.

guide” will explain in ADELFE) allows the determination of the relevant adjustments to do.

VI. CONCLUSION

In this article a first step was made towards integrating a methodological guide into ADELFE for helping engineers when designing the behaviour of cooperative agents. This preliminary textual guide may be summed up as follows. The designer has first to devise the nominal behaviour of agents involved in the AMAS he has to engineer. Depending on the feedbacks given by end-users, he has then to act on the adaptive behaviour of these agents to improve the collective behaviour of this AMAS. For the time being, only the tuning part of this adaptive behaviour was studied. The designer has then to establish relationships between parameters involved in the AMAS and actions agents may do (and this may be done

before, as a help for finding the nominal behaviour also). Then, these relationships have to be quantified and studied, especially by propagating effects a parameter may have on different actions (promote or deserve them). This enables finding which and how some parameters have to be changed for positively influencing the collective behaviour towards what is expected by end-users.

A MAS simulating cancerous cell proliferation in a tissue was used for grounding this demonstration. Although it cannot be considered as having a strong biological reality, it accounts for an interesting complex use case, at least for tuning, because it has several interrelated actions and parameters. The proposed approach was applied on this cell simulation for playing on the evolution of the ratio of cancerous cells over time and trying to influence the shape of the related curve. By changing some parameters used by healthy and cancerous cells, this curve was actually changed in the right direction. As a consequence the distribution of cells in the tissue was also modified which shows that emergence of a phenomenon at the global level may be influenced by changes in the local behaviours.

However, a lot of work has to be done yet. First, it is necessary to study how reorganisation and evolution behaviours could be used for guiding the engineer, then enriching ADELFE with this (still textual) guide could be done, once formalised. The last step would be to automate this guide by implementing the related tool as an AMAS. Indeed, discovering the parameters and actions to adjust in a complex adaptive system clearly corresponds to a specific agentification level, not directly required by the end-user problem. In this tool, for at least the tuning part, actions and parameters would be considered as cooperative agents. Their collective goal would be to find the right parameters to modify depending on the feedbacks given by end-users which would be considered as NCS that these agents should avoid and solve. Furthermore, all the results observable by an end-user would have to be agentified (e.g. the curve shown on Fig. 2 would be an agent in order to reason cooperatively on function R when an external feedback occurs). Consequently, engineering semi-automatically the development of a complex adaptive system must be AMAS-compliant, even when this complex system itself is not AMAS-designed.

Consequently this tool is not ADELFE-dependent and could be included in any methodology devoted to the development of complex adaptive systems. Nevertheless, a lot of work and tests are required before considering such a kind of deployment.

REFERENCES

- [1] F. Bergenti, M.-P. Gleizes, and F. Zambonelli, *Methodologies and Software Engineering for Agent Systems*. Kluwer Publishing, 2004.
- [2] B. Henderson-Sellers and P. Giorgini, *Agent-Oriented Methodologies*. Idea Group Pub, June 2005.
- [3] D. Capera, J.-P. Georgé, M.-P. Gleizes, and P. Glize, "The AMAS Theory for Complex Problem Solving Based on Self-organizing Cooperative Agents," in *12th IEEE International Workshops on Enabling Technologies, Infrastructure for Collaborative Enterprises, Linz, Austria*. IEEE Computer Society, June 2003, pp. 383–388.
- [4] C. Bernon, D. Capera, and J.-P. Mano, "Engineering Self-Modeling Systems: Application to Biology," in *International Workshop on Engineering Societies in the Agents World*, ser. LNCS, A. Artikis, G. Picard, and L. Vercoeur, Eds., no. 5485. Springer, 2009, pp. 236–251.
- [5] A. Uhrmacher, "Simulation for Agent-Oriented Software Engineering," in *First International Conference on Grand Challenges for Modelling and Simulation, San Antonio, Texas*, W. Lunceford and E. Page, Eds. San Diego: SCS, 2002.
- [6] G. Fortino, A. Garro, W. Russo, R. Caico, M. Cossentino, and F. Termine, "Simulation-Driven Development of Multi-Agent Systems," in *Workshop on Multi-Agent Systems and Simulation, Palermo, Italia*, 2006.
- [7] M. Cossentino, G. Fortino, A. Garro, S. Mascillaro, and W. Russo, "PASSIM : A Simulation-based Process for the Development of Multi-Agent Systems," *Int. Journal of Agent-Oriented Software Engineering*, vol. 2, no. 2, pp. 132–170, 2008.
- [8] L. Gardelli, M. Viroli, M. Casadei, and A. Omicini, "Designing Self-organising Environments with Agents and Artefacts: A Simulation-driven Approach," *Int. J. Agent-Oriented Softw. Eng.*, vol. 2, no. 2, pp. 171–195, 2008.
- [9] C. Bernon, M.-P. Gleizes, and G. Picard, "Enhancing Self-Organising Emergent Systems Design with Simulation," in *International Workshop on Engineering Societies in the Agents World (ESAW), Dublin*, ser. LNCS, G. O'Hare, M. O'Grady, A. Ricci, and O. Dikenelli, Eds., vol. 4457. Springer-Verlag, September 2007, pp. 284–299.
- [10] S. Lemouzy, C. Bernon, and M.-P. Gleizes, "Living Design: Simulation for Self-Designing Agents," in *European Simulation and Modelling Conference (ESM), Malte*, J. Sklenar, A. Tanguy, C. Bertelle, and G. Fortino, Eds. Eurosis, October 2007, pp. 432–436.
- [11] F. Klügl, R. Herrler, and O. Oechslein, "From Simulated to Real Environments: How to use SeSAM for Software Development," in *Multiagent System Technologies - 1st German Conference MATES*, ser. LNAI, M. Schillo, Ed., vol. 2831. Springer-Verlag, 2003, pp. 13–24.
- [12] T. Fisher and T. Henzinger, "Executable Cell Biology," *Nature Biotechnology*, vol. 25, no. 11, pp. 1239–1249, November 2007.
- [13] P. Mendes, "GEPASI: A Software Package for Modelling the Dynamics, Steady States and Control of Biochemical and other Systems," *Computer App. in the Biosciences*, vol. 9, no. 5, pp. 563–571, October 1993.
- [14] C. Li, Q. W. Ge, M. Nakata, H. Matsuno, and S. Miyano, "Modelling and Simulation of Signal Transductions in an Apoptosis Pathway by using Timed Petri Nets," *J. Biosci.*, vol. 32, pp. 113–127, 2007.
- [15] H. Silva and M. Martins, "A Cellular Automata Model for Cell Differentiation," *Physica A*, no. 322, pp. 555–566, 2003.
- [16] E. Merelli, G. Armano, N. Cannata, F. Corradini, M. d'Inverno, A. Doms, P. Lord, A. Martin, L. Milanese, S. Moller, M. Schroeder, and M. Luck, "Agents in bioinformatics, Computational and Systems Biology," *Briefings in Bioinformatics*, vol. 8, no. 1, pp. 45–59, 2006.
- [17] M. d'Inverno and R. Saunders, "Agent-Based Modelling of Stem Cell Self-organisation in a Niche," in *Engineering Self-Organising Systems: Methodologies and Applications*, ser. LNCS, S. Brueckner, G. Di Marzo Serugendo, A. Karageorgos, and R. Nagpal, Eds., vol. 3464. Springer, 2004, pp. 52–68.
- [18] L. Dib, N. Bonnet, Z. Guessoum, and M. Laskri, "Multi-agent System Simulating Tumoral Cells Migration," in *18th Australian Joint Conference on Artificial Intelligence*. Springer Verlag, December 2005, pp. 624–632.
- [19] K. Webb and T. White, "Cell Modeling with Reusable Agent-based Formalisms," *Applied Intelligence*, vol. 24, no. 2, pp. 169–181, April 2006.
- [20] T. J. Weber, R. W. Siegel, L. M. Markillie, W. B. Chrisler, X. C. Lei, and N. H. Colburn, "A Paracrine Signal Mediates the Cell Transformation Response to Low Dose Gamma Radiation in JB6 Cells," *Molecular Carcinogenesis*, no. 43, pp. 31–37, 2005.

Simulation of Alternative Self-Organization Models for an Adaptive Environment

Stefania Bandini, Andrea Bonomi, Giuseppe Vizzari
Complex Systems and Artificial Intelligence research centre
University of Milano–Bicocca
Viale Sarca 336, U14 Building, 20126 Milano, Italy
{bandini, bonomi, vizzari}@disco.unimib.it

Vito Acconci
Acconci Studio
20 Jay St., Suite #215, Brooklyn, NY 11201
studio@acconci.com

Abstract—The ambient intelligence scenario depicts electronic environments that are sensitive and responsive to the presence of people. The paper deals with a particular kind of system whose aim is to enhance the everyday experience of people moving inside the related physical environment according to the narrative description of a designer's desiderata. In this kind of situation computer simulation represents a useful way to envision the behaviour of the responsive environments modeled and implemented, without actually bringing them into existence in the real world, in order to evaluate their adherence to the designer's specification. This paper describes the simulation of an adaptive illumination facility, a physical environment endowed with a set of sensors that perceive the presence of humans (or other entities such as dogs, bicycles, cars) and interact with a set of actuators (lights) that coordinate their state to adapt the ambient illumination to the presence and behaviours of its users. The simulation system is used to compare two different self-organization models managing the adaptive illumination system.

I. INTRODUCTION

The ambient intelligence scenario [16] describes future human environments as dynamic places, endowed with a large number of wirelessly interconnected electronic devices that sense the nearby conditions and react to the perceived signals. The aims of these facilities can be very different, from explicitly providing electronic services to humans present in the environment through some form of computational device (such as personal computer or PDA), to simply realizing some type of ambient adaptation to the users' presence (or deliberate acts like voice emission or gestures). Ambient intelligence comprises thus those systems that are designed to autonomously adapt the environment to the people living or simply passing by in it in order to improve their everyday experience.

Sometimes the requirements and the specification of this form of adaptation are clear, unambiguous and even already formalized (e.g. in the wintertime keep the internal temperature of each room between 19 C and 22 C); on the other hand, sometimes the idea and specification of the desired adaptation is given in a visual or narrative form by a designer or even by an artist (in case of artistic installations). In this case, while the desired overall effect of adaptation could be clear it may be very complex to fill the gap between this form of high level specification and a computational system.

In this second situation computer simulation can play a crucial role in supporting the design and realization of adaptive, self-organizing ambient intelligence systems. In fact, traditional design and modeling instruments can provide a suitable support for evaluating static properties of this kind of environment (e.g. through the construction of 3D models representing a mock-up, proof of concept of the desired appearance or also adaptation effect but in a single specific situation), but they are not designed to provide abstractions and mechanisms for the definition and simulation of reactive environments and their behaviours. Through the definition of specific models and their implementation in simulators it is possible to obtain an envisioning of the static features of the ambient intelligence system as well as its dynamic response to the behaviour of humans and other relevant entities situated in it. This allows performing a *face validation* [14] of the adaptation mechanisms and also to perform a tuning of the relevant parameters.

This paper describes the application of a modeling and simulation approach to support the design of an adaptive illumination facility that is being designed and realized by the Acconci Studio¹ in Indianapolis. In particular, the designed system should be able to locally enhance the overall illumination of a tunnel in order to highlight the position and close surrounding area of pedestrians (as well as other entities such as dogs, bicycles, cars). In this case, the simulation offers both a support to the decisions about the number and positioning of lights and, more important, it encapsulates the self-organization mechanisms guiding the adaptive behaviour of lights reacting to the presence of pedestrians and other relevant entities in the environment. By providing the current state of the environment, in terms of simulated outputs of sensors detecting the presence of pedestrians, as an input to the self-organization model it is possible to obtain its simulated response, and the current state of lights. A schema of the overall simulation system is shown in Figure 1: it must be noted that the self-organization model adopted for the simulator could be effectively used to manage the actual system, simply providing actual inputs from field sensors and employing its outputs to manage actual lights rather than a

¹<http://www.acconci.com>

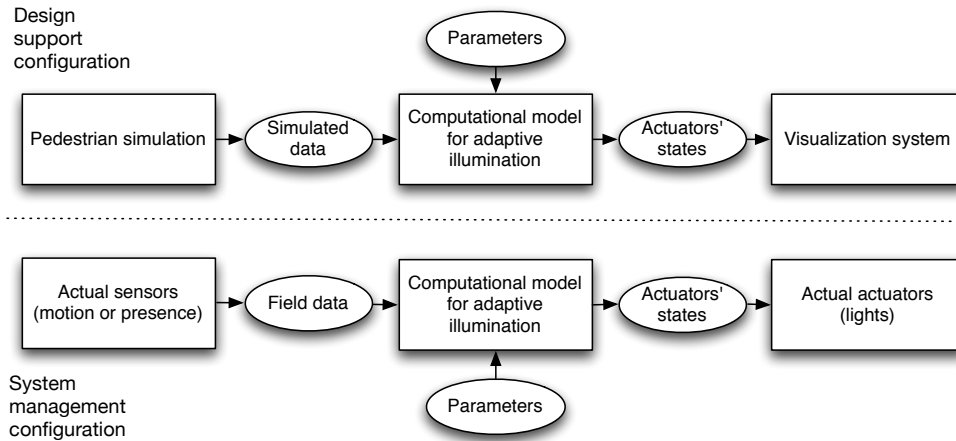


Fig. 1. A schema describing the components of the simulation system.

virtual visualization of the actual environment.

Besides the specific aims of the ambient intelligent system, there is an increasing interest and number of research efforts on approaches, models and mechanisms supporting forms of self-organization and management of the components (both hardware and software) of such systems. The latter are growingly viewed in terms of autonomous entities, managing internal resources and interacting with surrounding ones so as to obtain the desired overall system behaviour as a result of local actions and interactions among system components. Examples of this kind of approach can be found in both in relatively traditional pervasive computing applications (see, e.g., [10]), but also in a new wave of systems developed in the vein of amorphous computing [2] such as the one on paintable computers described in [9]. In this rather extreme application a whole display architecture is composed of autonomous and interacting graphic systems, each devoted to a single pixel, that must thus interact and coordinate their behaviours even to display a simple character. There is however a significant number of heterogeneous approaches to the definition of models supporting forms of self-organization in artificial systems and their application involves several important modeling and engineering choices.

It must be stressed that in the Indianapolis tunnel renovation scenario the designer had a precise idea of the desired overall adaptive illumination effect, combining a functional overall illumination of the tunnel – allowing pedestrians and drivers to effectively have a sufficient visibility of the environment but also emphasizing their presence and passage through the tunnel – but the choice of the computational model to achieve this effect was definitely not obvious. For instance, the rationale of the desired adaptive illumination pattern (that will be more thoroughly described in Section II) is to manage lights as if they were animated and able to follow the movement of pedestrians. This was technically impossible in the specific scenario and the effect had to be achieved by turning on and off in a coordinated way a set of lights characterized by a fixed

position in the environment. However, the metaphor adopted by the designer could lead to consider specific computational models whose first class concepts are moving entities, like Boids [15], whose presence in the portion of the environment associated to a light indicates the need to turn a specific light on. On the other hand, a different approach, based on Cellular Automata [17], would lead to consider the physical structure of the environment, that is, a discrete and finite grid whose nodes could be either sensors or actuators, and the effective nature of the kind of action that must be managed, that is a change of state of the actuators.

The aims of the paper are thus twofold: on one hand it describes a concrete experience in which computer simulation was adopted to fill the gap between an abstract specification of the desired behaviour of an adaptive self-organizing environment; on the other hand the paper discusses the adequacy and feasibility of the adoption of two alternative computational models to generate the desired adaptation effect and to be effectively deployed in the real infrastructure.

The following section will introduce more in details the specific scenario in which this research effort is set, describing the requirements for the adaptive illumination system and the environment adaptation model. Section III introduces the pedestrian modeling approach, while the self-organization models that were experimented to guide the adaptive illumination facility are described in Section IV. A description of the developed environment supporting designers will follow, then conclusions and future works will end the paper.

II. THE SCENARIO

The Acconci Studio, partner of the described research effort, has recently been involved in a project for the renovation of a tunnel in the Virginia Avenue Garage in Indianapolis. The tunnel is currently mostly devoted to cars, with relatively limited space on the sidewalks and its illumination is strictly functional. The planned renovation for the tunnel includes a set of interventions, and in particular two main effects of



Fig. 2. A visual elaboration of the desired adaptive illumination facility (the image appears courtesy of the Acconci Studio).

illumination, also depicted in a graphical elaboration of the desired visual effect shown in Figure 2: an overall effect of *uniformly coloring* the environment through a background, ambient light that can change through time, but slowly with respect to the movements and immediate perceptions of people passing in the tunnel; a *local effect of illumination* reacting to the presence of pedestrians, bicycles, cars and other physical entities.

The rationale of this local and dynamic adaptive illumination effect is better explained by the following narrative description of the desired effect:

The passage through the building should be a volume of color, a solid of color. It's a world of its own, a world in itself, separate from the streets outside at either end. Walking, cycling, through the building should be like walking through a solid, it should be like being fixed in color.

The color might change during the day, according to the time of day: pink in the morning, for example, becomes purple at noon becomes blue, or blue-green, at night. This world-in-itself keeps its own time, shows its own time in its own way.

The color is there to make a heaviness, a thickness, only so that the thickness can be broken. The thickness is pierced through with something, there's a sparkle, it's you that sparkles, walking or cycling though the passage, this tunnel of color. Well no, not really, it's not you: but it's you that sets off the sparkle – a sparkle here, sparkle there, then another sparkle in-between – one sparkle affects the other, pulls the other, like a magnet – a point of sparkle is stretched out into a line of sparkles is stretched out into a network of sparkles.

These sparkles are above you, below you, they spread out in front of you, they light your way through the tunnel. The sparkles multiply: it's you

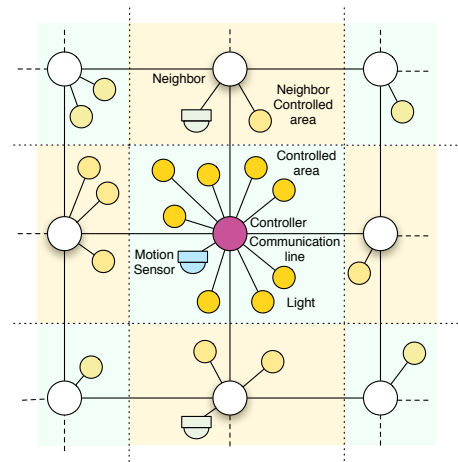


Fig. 3. A schema of the CA model for the adaptive illumination facility.

who sets them off, only you, but – when another person comes toward you in the opposite direction, when another person passes you, when a car passes by – some of these sparkles, some of these fire-flies, have found a new attractor, they go off in a different direction.

The first type of effect can be achieved in a relatively simple and centralized way, requiring in fact a uniform type of illumination that has a slow dynamic. The second point requires a different view on the illumination facility. In particular, it must be able to perceive the presence of pedestrians and other physical entities passing in it, in other words it must be endowed with sensors (detecting either the presence or the movement of relatively big objects). Moreover, it must be able to exhibit local changes as a reaction to the outputs of the aforementioned sensors, providing thus for a non uniform component to the overall illumination. The overall environment must be thus split into parts, cells that represent proper subsystems: Figure 3 shows a schema of the approach we adopted to subdivide the physical environment into autonomous units, provided with motion/presence sensors (able to detect the arrival/presence of relevant entities) and lights (to adapt the ambient illumination, highlighting the presence of pedestrians).

However, the effect of the presence of a pedestrian in a portion of space should extend beyond the borders of the occupied cell. In fact, the illumination effect should “light the way” of a pedestrian through the tunnel. Cells must thus be able to interact, in order to influence neighboring ones whenever a pedestrian is detected, to trigger a (maybe less intense) illumination.

III. PEDESTRIAN SIMULATION MODEL

The adopted pedestrian model is based on the Situated Cellular Agent model, a specific class of Multilayered Multi-Agent Situated System (MMASS) [6] providing a single layered spatial structure for agents environment. A thorough description of the model is out of the scope of this paper, but we briefly introduce it to give some basic notion of

the elements that are necessary to describe the SCA crowd modeling approach.

A. Situated Cellular Agents

A *Situated Cellular Agent* system is defined by the triple $\langle \text{Space}, F, A \rangle$ where *Space* models the environment where the set A of agents is situated, acts autonomously and interacts through the propagation of the set F of fields and through reaction operations. *Space* consists of a set P of sites arranged in a network (i.e. an undirected graph of sites). The structure of the space can be represented as a neighborhood function, $N : P \rightarrow 2^P$ so that $N(p) \subseteq P$ is the set of sites adjacent to $p \in P$; the previously introduced *Space* element is thus the pair $\langle P, N \rangle$. Focusing instead on the single basic environmental elements, a site $p \in P$ can contain at most one agent and is defined by the 3-tuple $\langle a_p, F_p, P_p \rangle$ where:

- $a_p \in A \cup \{\perp\}$ is the agent situated in p ($a_p = \perp$ when no agent is situated in p that is, p is empty);
- $F_p \subset F$ is the set of fields active in p ($F_p = \emptyset$ when no field is active in p);
- $P_p \subset P$ is the set of sites adjacent to p (i.e. $N(p)$).

A SCA agent is defined by the 3-tuple $\langle s, p, \tau \rangle$ where τ is the *agent type*, $s \in \Sigma_\tau$ denotes the *agent state* and can assume one of the values specified by its type (see below for Σ_τ definition), and $p \in P$ is the site of the *Space* where the agent is situated. As previously stated, *agent type* is a specification of agent state, perceptive capabilities and behaviour. In fact an agent type τ is defined by the 3-tuple $\langle \Sigma_\tau, \text{Perception}_\tau, \text{Action}_\tau \rangle$. Σ_τ defines the set of states that agents of type τ can assume. $\text{Perception}_\tau : \Sigma_\tau \rightarrow [\mathbf{N} \times W_{f_1}] \dots [\mathbf{N} \times W_{f_{|F|}}]$ is a function associating to each agent state a vector of pairs representing the *receptiveness coefficient* and *sensitivity thresholds* for that kind of field. Action_τ represents instead the behavioural specification for agents of type τ . Agent behaviour can be specified using a language that defines the following primitives:

- $\text{emit}(s, f, p)$: the *emit* primitive allows an agent to *start the diffusion of field f* on p , that is the site it is placed on;
- $\text{react}(s, a_{p_1}, a_{p_2}, \dots, a_{p_n}, s')$: this kind of primitive allows the specification of a *coordinated change of state* among adjacent agents. In order to preserve agents' autonomy, a compatible primitive must be included in the behavioural specification of all the involved agents; moreover when this coordination process takes place, every involved agents may dynamically decide to effectively agree to perform this operation;
- $\text{transport}(p, q)$: the *transport* primitive allows to *define agent movement* from site p to site q (that must be adjacent and vacant);
- $\text{trigger}(s, s')$: this primitive specifies that an agent must *change its state* when it senses a particular condition in its local context (i.e. its own site and the adjacent ones); this operation has the same effect of a reaction, but does not require a coordination with other agents.

For every primitive included in the behavioural specification of an agent type specific preconditions must be specified; moreover specific parameters must also be given (e.g. the specific field to be emitted in an emit primitive, or the conditions to identify the destination site in a transport) to precisely define the effect of the action, which was previously briefly described in general terms.

Each SCA agent is thus provided with a set of sensors that allows its interaction with the environment and other agents. At the same time, agents can constitute the source of given fields acting within a SCA space (e.g. noise emitted by a talking agent). Formally, a field type t is defined by $\langle W_t, \text{Diffusion}_t, \text{Compare}_t, \text{Compose}_t \rangle$ where W_t denotes the set of values that fields of type t can assume; $\text{Diffusion}_t : P \times W_f \times P \rightarrow (W_t)^+$ is the diffusion function of the field computing the value of a field on a given space site taking into account in which site (P is the set of sites that constitutes the SCA space) and with which value it has been generated. It must be noted that fields diffuse along the spatial structure of the environment, and more precisely a field diffuses from a source site to the ones that can be reached through arcs as long as its intensity is not voided by the diffusion function. $\text{Compose}_t : (W_t)^+ \rightarrow W_t$ expresses how fields of the same type have to be combined (for instance, in order to obtain the unique value of field type t at a site), and $\text{Compare}_t : W_t \times W_t \rightarrow \{\text{True}, \text{False}\}$ is the function that compares values of the same field type. This function is used in order to verify whether an agent can perceive a field value by comparing it with the sensitivity threshold after it has been modulated by the receptiveness coefficient.

B. SCA Based Pedestrian Model

The above introduced SCA model has been applied to represent a very simple tunnel with two ends and some columns in it; pedestrians enter the tunnel from one end and they move towards the other end, avoiding obstacles either immobile (i.e. columns), and mobile (i.e. other pedestrians moving in the opposite direction).

The SCA *Space* is the same cellular space defined for the D-MAN described in Section IV. To support agent navigation in this space, in each end of the tunnel we positioned an additional site in which a “beacon” agent (a static agent emitting a simple presence field) is situated. In the environment, thus, only two types of field are present.

To exploit this environmental specification in order to obtain the above overall system behaviour, we defined two types of agent, respectively interpreting the one type of field as attractive and ignoring the other one. This can be achieved through a simple *transport* primitive, specifying that the agent should move towards the free adjacent site in which the intensity of the field considered attractive is maximum. The behavioural specification of these agents is completed by an obstacle avoidance rule (another *transport* that moves the agent towards a random different lane whenever the best possible destination is occupied by an obstacle). Finally, agents reaching their destination, that is, one of the tunnel ends, are

removed from the environment and they are positioned at the other end, so they start over their crossing of the tunnel.

IV. ADAPTIVE ILLUMINATION MODEL

A. CA Based Approach

We employed a Cellular Automata model to realize the local effect of illumination as a self-organized reaction to the presence of pedestrians. CA cells, related to a portion of the physical environment, comprise sensors and actuators, as schematized in Figure 3. The former can trigger the behaviours of the latter, both through the interaction of elements enclosed in the same cell and by means of the local interaction among adjacent cells. The transition rule models mechanisms of reaction and diffusion, and it was derived by previous applications to reproduce natural phenomena such as percolation processes of pesticides in the soil, in percolation beds for the coffee industry and for the experimentation of elasticity properties of batches for tires [5]. In this specific application the rule manages the interactions of cells arranged through a multilayered architecture based on the Multilayered Automata Network model [8], schematized in Figure 3.

Multilayered Automata Network have been defined as a generalization of Automata Networks [11]. The main feature of the Multilayered Automata Network is the explicit introduction of a hierarchical structure based on nested graphs, that are graphs whose vertexes can be in turn be a nested graph of lower level. A Multilayered Automata Network is directly obtained from the nested graph structure by introducing states and a transition function.

The irregular nature of the cellular space is not the only difference between the adopted approach and the traditional CA models. In fact, CAs are in general closed and synchronous systems, in which cells update their state in parallel triggered by a global clock. Dissipative Cellular Automata (DCA) [18] differ from the basic CAs mainly for two characteristics: while CA are synchronous and closed systems, DCA are open and asynchronous. DCA cells are characterized by a thread of control of their own, autonomously managing the elaboration of the local cell state transition rule. DCA can thus be considered as an open agent system [13], in which the cells update their state independently of each other and they are directly influenced by the environment.

The model we defined and adopted, Dissipative Multilayered Automata Network (D-MAN), takes thus the advantages of both the Multilayered Automata Network and the Dissipative Cellular Automata. An informal definition this model describes D-MAN as Multilayered Automata Network in which the cells update their state in an asynchronous way and they are open to influences by the external environment.

The multilayered cellular structure of the D-MAN is composed of three layers:

- the first layer is related to the basic *discretization* of the physical environment into cells, corresponding to a local controller. Each of these cells effectively comprises the two additional layers;

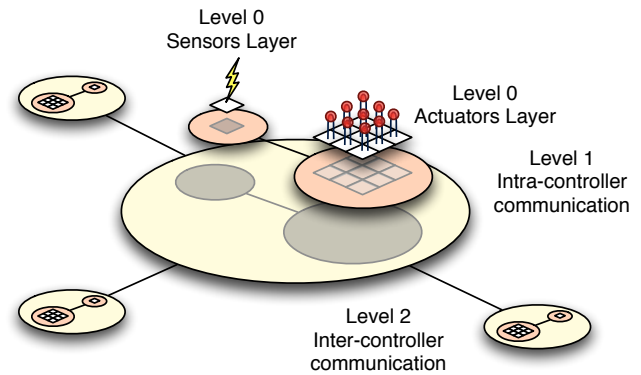


Fig. 4. The proposed automata network for the D-MAN.

- the *perception* and *actuation* layers, respectively comprising the sensors and actuators (lights).

This structure is schematized in Figure 4. The rationale of keeping separated these cells is to be able to specify and configure specific functions describing (i) how to compute the overall internal activation state of a cell given the status of the internal sensor(s) and the current state of activation of neighbours and (ii) how to translate this state of activation into a state of actuation for that specific layer (in other words, how to translate into a lighting effect the state of the cell).

Specific transition rules must thus be defined to manage different interactions and influences that take place in this structure, and mainly (i) the direct influence of a sensor that detected a pedestrian to the actuators in the same cell, and (ii) the influence of a high level cell to the neighboring ones (given the internal structure of each cell, due to the presence of a specific level of actuators inside it, this interaction effectively affects *a part* of a neighboring cell). Moreover, the effect of external stimuli must gradually vanish, and lights must fade in absence of pedestrians: while an active state of the sensor and high activation states of neighbours cause an increase of the cell activation state, it decreases in absence of these triggering conditions. More details about the formal definition of the model can be found in [4].

The adaptive illumination model is thus characterized by several features that make it difficult to predict how it will react to particular stimuli (i.e. patterns of pedestrian movement in the related environment), from the number and positioning of sensors and actuators, to the parameters of the transition rule. The transition rule per se is characterized by several parameters whose configuration can actually deeply alter the achieved illumination effect, to the point that we developed an ad hoc UI to show its behaviour when triggered through mouse clicks [3]. To couple this model with a pedestrian simulation model sharing the discrete representation of the spatial aspect of the environment allows to simulate the behaviour of the adaptive illumination facility as a response to specific patterns of usage of the environment by pedestrians was thus considered an effective way to envision the performance of the adaptive illumination facility in plausible situations.

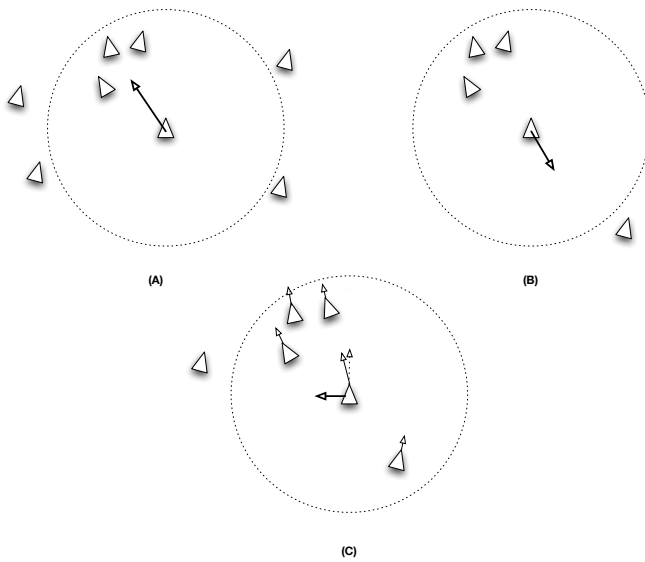


Fig. 5. A diagram schematizing the three basic steering behaviours of a boid: (A) cohesion, (B) separation and (C) alignment.

B. Boids Based Approach

The narrative description of the desired adaptive illumination effect by the designer explicitly mentioned “fire-flies”, insects forming a swarm. This description effectively leads to consider the possibility to adopt for the self-organization of the illumination facility computational models developed to generate collective behaviours of insects (but also flocks, herds and schools) such as the *boids* model. This particular model is very effective in generating coordinated animal motion, with a relatively simple computation essentially based on the mutual local perception of individuals situated in a physical environment. In our system, lights cannot actually move, but boids can represent the fact that a light is active in a predefined environment structure representing a discrete and finite set of allowed positions. The movement of boids represents thus the dynamic update of lights’ statuses in the illumination facility.

The basic model comprises three simple steering behaviours, describing how an individual boid maneuvers basing on the positions and velocities its nearby flockmates. These behaviours, also schematized in Figure 5, are:

- *cohesion*: steer to move toward the average position (the centroid) of local flockmates;
- *separation*: steer to avoid crowding (i.e. in the opposite direction of the the centroid of local flockmates);
- *alignment*: steer towards the average heading of local flockmates.

The notion of locality mentioned in these rules is essentially related to the possibility of boids to detect others of their kind situated within a certain range from their current position. These three steering behaviours produce vector representing a contribution to the overall action of the single boid, that is thus obtained as the vectorial sum of these contributions multiplied for specific scalar constants. These constants must

be properly calibrated to avoid excessive dispersion and cohesion of the boids. Moreover, in this specific application, boids must also be attracted by people situated in the environment: a fourth contribution to the overall boid behaviour must thus be introduced, otherwise the boids wander in a realistic way (from the point of view of simulating a collective behaviour) but completely ignoring the presence of humans in the environment.

Before introducing an additional contribution to the behavioural specification of a boid to tackle this issue, we first considered that the basic boid model is conceived for a continuous environmental spatial representation, that is not suited to this situation. The adopted approach was to translate the cohesion and separation contributions of the model into elements of the aforementioned SCA model, mainly for two reasons: (i) this model is by definition discrete and (ii) it was already adopted to model pedestrians and the environment they are situated in. Alignment was not considered since SCA agents are not characterized by a direction or heading in space, but just by their position (i.e. a node in a graph structure).

To realize a model featuring the main characteristics of boids using SCAs we decided to provide each boid with a form of *presence* field, diffusing a sign indicating its position in nearby sites. Presence fields’ diffusion function decreases by a constant value the intensity of a field for each node the site crosses in the course of the diffusion operation until the value is void, while the composition function simply adds up the value of all presence fields in a given site. In this way, presence field in a given site represents a measure of its crowdedness and it can be used by boids to select sites that represent a good compromise between cohesion and separation. The behavioural specification of SCA boids comprises thus essentially two basic actions, a field emission and a transport action interpreting the value of the presence field as sort of social force [12]. In addition, we model the presence of pedestrians as another presence fields that is generated by the pedestrian agents present in the coordinated model simulating the behaviours of people moving in the tunnel. Boids agents’ transport action favors thus as a preferred destination those sites characterized by an average intensity of the boids presence field and a high intensity for pedestrian presence field. Also in this case, the model is characterized by a number of parameters having a serious impact on the overall adaptive illumination effect.

C. Discussion

The above introduced models were adopted and tested in the Indianapolis tunnel scenario; both of them proved their adequacy to effectively represent a formal, computational, non ambiguous and executable specification of the designer’s narration. They undergone a successful face validation, that followed several iterations to define a good value for the models’ parameters to achieve the desired results. However, the models have specific features that can have an impact on some of the non-functional properties they exhibit.



Fig. 6. Screenshot of the simulation environment: the central window shows a three-dimensional view of the tunnel (the actual 3D model of the tunnel was adopted), while the top panel shows a bi-dimensional view highlighting the position of pedestrians and the state of lights.

It must be stressed that even if the designer used the term “fire-flies” in the narrative description of the desired effect, the idea to follow the metaphor and to employ one of the most commonly adopted model for this kind of collective behaviour does not necessarily imply a smooth and simple definition of a model achieving the desired effect. First of all the boids model is based on a continuous spatial representation and thus it must be adapted to the discrete spatial structure of the illumination facility. Then a modification to the basic model must be introduced to achieve a “goal driven” behaviour (i.e. the tendency of boids to move towards people, preserving the swarm behaviour). Finally it must be noted that this modeling approach does not take into account the effective infrastructure that will be effectively employed to realize the illumination facility.

The CA based approach is based on the idea of viewing the environment itself as an assembly of autonomous units able to interact with their neighbours. The adaptive illumination effect is achieved as a reaction of these units to an external stimulus generated by sensors and as a results of their interaction. As a result it is much simpler to conceive a direct effective implementation of this approach in a concrete system made up of a set of micro-controllers responsible for the monitoring of a certain part of the environment and for the control of

the lights it includes. On the other hand, there is no simple way to distribute the boids model in a distributed control system: the simple fact that boids can perceive the presence of other individuals of their kind in a potentially distant position, according to their range of perception, leads to consider that in case of distribution of this form of computation to different autonomous units would lead to higher costs in terms of network communication, that is instead essentially constant in the CA based case.

V. CONCLUSIONS AND FUTURE WORKS

The paper introduced a simulation approach to supporting the design of an ambient intelligence infrastructure aimed at improving the everyday experience of pedestrians and people passing through the related environment. A specific scenario related to the definition and development of an adaptive illumination facility was introduced, and two different computational models model specifying its dynamic behaviour was defined. An agent-based pedestrian model simulating inputs and stimuli to the adaptation module was also introduced. The models described in the previous Sections are part of a prototype supporting the design and definition of the overall illumination facility through the simulation and envisioning of its dynamic behaviour according to specific values for the relevant parameters (e.g. parameters of the transition rule of

the CA, but also the number of lights and sensors, and so on); a screenshot of the system is shown in Figure 6: the central window shows a three-dimensional view of the tunnel, while the top panel shows a bi-dimensional view highlighting the position of pedestrians and the state of lights. The simulation takes place in the actual 3D model of the tunnel, that was adopted to achieve in a semi-automatic way a discrete representation of the environment that was adopted both for enabling the pedestrian simulation and for the positioning of lights. Part of the simulator, based on an platform for agent-based simulation [7], generates patterns of movement of pedestrians simulating inputs for the CA and another part of the system generates a visualization of the system dynamics, interpreting the states of the CA.

The renovation project is currently under development on the architectural and engineering side, whereas the introduced models have shown their adequacy to the problem specification, both in order to provide a formal specification of the behaviour for the system components and possibly as a control mechanism. The realized prototype explored the possibility of realizing an ad hoc tool that can integrate the traditional CAD systems for supporting designers in simulating and envisioning the dynamic behaviour of complex, self-organizing installations. It has been used to understand the adequacy of the modeling approach in reproducing the desired self-organized adaptive behaviour of the environment to the presence of pedestrians. We are currently improving the prototype, on one hand, to provide a better support for the Indianapolis project and, on the other, to realize a more general framework for supporting designers of dynamic self-organizing environments.

ACKNOWLEDGEMENTS

The authors wish to thank some collaborators at the Acconci Studio and in particular Nathan DeGraaf, Jono Podborseck and James Clar for their collaboration to the present work and research effort.

REFERENCES

- [1] *Proceedings of the First International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2007, Boston, MA, USA, July 9-11, 2007*. IEEE Computer Society, 2007.
- [2] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. F. K. Jr., R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss. Amorphous computing. *Commun. ACM*, 43(5):74–82, 2000.
- [3] S. Bandini, A. Bonomi, G. Vizzari, and V. Acconci. An Asynchronous Cellular Automata-based Adaptive Illumination Facility. In *Proceedings of the Ninth Conference of the Italian Association for Artificial Intelligence*, page (to appear), 2009.
- [4] S. Bandini, A. Bonomi, G. Vizzari, V. Acconci, N. DeGraaf, J. Podborseck, and J. Clar. A CA-Based Approach to Self-Organized Adaptive Environments: The Case of an Illumination Facility. In *Self-Adaptive and Self-Organizing Systems Workshops, 2008. SASOW 2008. Second IEEE International Conference on*, pages 1–6, Oct. 2008.
- [5] S. Bandini, G. Erbacci, and G. Mauri. Implementing Cellular Automata Based Models on Parallel Architectures: The CAPP Project. In V. E. Malyskin, editor, *PaCT*, volume 1662 of *Lecture Notes in Computer Science*, pages 167–179. Springer, 1999.
- [6] S. Bandini, S. Manzoni, and C. Simone. Dealing with space in multi-agent systems: a model for situated MAS. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1183–1190. ACM Press, 2002.
- [7] S. Bandini, S. Manzoni, and G. Vizzari. Towards a Platform for Multilayered Multi Agent Situated System Based Simulations: Focusing on Field Diffusion. *Applied Artificial Intelligence*, 20(4–5):327–351, 2006.
- [8] S. Bandini and G. Mauri. Multilayered Cellular Automata. *Theor. Comput. Sci.*, 217(1):99–113, 1999.
- [9] W. Butera. Text display and graphics control on a paintable computer. In *SASO [1]*, pages 45–54.
- [10] A. E. S. Filho, E. C. Lupu, N. Dulay, S. L. Keoh, K. P. Twidle, M. Sloman, S. Heeps, S. Strowes, and J. Sventek. Towards supporting interactions between self-managed cells. In *SASO [1]*, pages 224–236.
- [11] E. Goles and S. Martinez. *Neural and Automata Networks: Dynamical Behavior and Applications*. Kluwer Academic Publishers, 1990. ISBN 0-792-30632-5.
- [12] D. Helbing and P. Molnár. Social force model for pedestrian dynamics. *Phys. Rev. E*, 51(5):4282–4286, May 1995.
- [13] N. R. Jennings. On agent-based software engineering. *Artif. Intell.*, 117(2):277–296, 2000.
- [14] F. Klügl. A Validation Methodology for Agent-Based Simulations. In R. Menezes and M. Viroli, editors, *Symposium on Applied Computing*, pages 39–43. ACM Press, 2008.
- [15] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34, New York, NY, USA, 1987. ACM.
- [16] N. Shadbolt. Ambient Intelligence. *IEEE Intelligent Systems*, 18(4):2–3, 2003.
- [17] J. von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, 1966.
- [18] F. Zambonelli, M. Mamei, and A. Roli. What can cellular automata tell us about the behavior of large multi-agent systems? In A. F. Garcia, C. J. P. de Lucena, F. Zambonelli, A. Omicini, and J. Castro, editors, *SELMAS*, volume 2603 of *Lecture Notes in Computer Science*, pages 216–231. Springer, 2002.

Verification & Validation of Agent Based Simulations using the VOMAS (Virtual Overlay Multi-agent System) approach

Muaz A. Niazi, Amir Hussain and Mario Kolberg

Abstract—Agent Based Models are very popular in a number of different areas. For example, they have been used in a range of domains ranging from modeling of tumor growth, immune systems, molecules to models of social networks, crowds and computer and mobile self-organizing networks. One reason for their success is their intuitiveness and similarity to human cognition. However, with this power of abstraction, in spite of being easily applicable to such a wide number of domains, it is hard to validate agent-based models. In addition, building valid and credible simulations is not just a challenging task but also a crucial exercise to ensure that what we are modeling is, at some level of abstraction, a model of our conceptual system; the system that we have in mind. In this paper, we address this important area of validation of agent based models by presenting a novel technique which has broad applicability and can be applied to all kinds of agent-based models. We present a framework, where a virtual overlay multi-agent system can be used to validate simulation models. In addition, since agent-based models have been typically growing, in parallel, in multiple domains, to cater for all of these, we present a new single validation technique applicable to all agent based models. Our technique, which allows for the validation of agent based simulations uses VOMAS: a Virtual Overlay Multi-agent System. This overlay multi-agent system can comprise various types of agents, which form an overlay on top of the agent based simulation model that needs to be validated. Other than being able to watch and log, each of these agents contains clearly defined constraints, which, if violated, can be logged in real time. To demonstrate its effectiveness, we show its broad applicability in a wide variety of simulation models ranging from social sciences to computer networks in spatial and non-spatial conceptual models.

Index Terms—Agent-based Modeling and Simulation, Multiagent System, Verification, Validation, Agent Oriented Software Engineering

I. INTRODUCTION

VALIDATION of any simulation model is a crucial task [1, 2]. Simulations, however well-designed, are always only an approximation of the system and if it was so easy to build the actual system, the simulation approach would never

have been used [3]. Of all the simulation models, agent-based modeling and simulation paradigm has recently gained a lot of popularity by being applied to a very wide range of domains such as [4-9]. Validation of models typically requires experts to look at data or animation as errors and un-wanted artifacts can appear in the development of agent-based models [10]. However, because of the complex nature of agent-based models comprising of multiple interacting entities and the strong dynamics and frequent emergence patterns in the system, it can be hard to validate agent-based models in the same way as traditional simulation models.

In the case of agent-based simulations, it is even easier to fall into the trap of tweaking the variables, especially since occasionally, the inputs can tend to be quite numerous [11]. Because of the complex nature of agent based models and resulting emergence as shown in [12-14], coupled with an enormous variation possibility of the variables, the results of the simulation study can vary considerably by changing the range or even the step size of just one or two variables. Thus, it is vitally important to be able to validate the agent-based simulation. The problem however, comes from the grounds up since validation is not to be an after-thought; it needs to be initiated alongside at the start of the simulation study. Now, validation of agent based models can be quite a challenging task [15, 16]. One problem lies in the fact that validation typically requires SME (Subject Matter Experts) to analyze [3] the simulation data or animation for comparison with another system or model. However, because of appearance of complex phenomenon such as emergence of behavior, where one plus one is not necessarily two as it depends more on the two “ones” and the behavior of the addition operation as is the norm in complex systems as compared to complicated systems [17]. Thus it can be very difficult to be sure if the behavior that we are observing is truly representative of the actual system [18]. Also, it is important to note here that even models, which cannot be validated might have merit and use such as bookkeeping devices or as an aid in selling ideas or as a training aid or even as part of an automatic management system. In the social sciences literature and ACE (Agents in Computation Economics), empirical validation of agent-based models has been described in [19]. Alternate approaches to empirical validation are discussed in [20]. Replication of agent-based models has been considered very important by some authors and has been discussed in [21]. An approach of validation based on philosophical truth theories in simulations

Manuscript received.

M. A. Niazi, A. Hussain and M. Kolberg are with the Department of Computing Science and Mathematics, University of Stirling, Scotland, UK. (Tel: +92-321-5310906; e-mail: {man, ahu, mko}@cs.stir.ac.uk).

has been discussed in [22]. Another approach called "companion modeling" is an iterative participatory approach where multidisciplinary researchers and stakeholders work together continuously throughout a four-stage cycle: field study and data analysis; role-playing games; agent-based model design and implementation; and intensive computational experiments [23]. Agent-based social simulation has also been used for validation and calibration [24].

In the past, although agent-based simulation has been shown to be useful in the validation of multi-agent systems[25, 26], multi-agent systems have not been used to validate agent-based models. On the other hand, simulations have been used in conjunction with software engineering for a long time[27]. Our work can be considered as pertaining to the last two stages of "Companion Modeling" i.e. Agent-Based Model Design/Implementation as well as Intensive Computational Experiments. Specifically, in this paper, we present the following innovations:

- We show how to develop a VOMAS (Virtual Overlay Multi-Agent System), which can be used for the validation of agent based simulation models.
- We thus further develop social science based validation techniques that can be applicable to both social science as well as other relevant domains.
- We present an object-oriented software engineering based methodology for validation of agent-based models, which provides for both logging as well as animation based validation approaches in addition to test-case/invariant based approaches.

The rest of the paper is structured as following: First we give an overview of the terms "Verification", "Validation" and "Credibility" as discussed in the literature. We also discuss how these terms have been considered traditionally in simulation models. Next, we give an overview of performing Validation using VOMAS. We show the design of VO (Virtual Overlay) and Logger agents. Next, we show an example of developing a VOMAS for an existing model from Agent-Based Modeling literature, and demonstrate its usefulness, and ease in validation. Finally we conclude the paper.

II. VERIFICATION, VALIDATION AND CREDIBILITY

Researchers transform real-world systems to models by applying abstraction. This transformation requires propagating concepts from the real world to useful computational models. These, in turn, are used to develop simulations. Simulation models, in essence end up giving back results which can be useful for the real world. As such, the more effective the abstraction mechanism, the better would be the expected real world benefits.

A. Peculiarities of Agent-Based Models

In case of agent-based models, the simulation comprises of one or more agents. These agents can work independently or

else interact with each other. These computational entities, which are typically, simplifications of real-world counterparts, need to have some meaningful semantics which can include anywhere from simple behaviors as well as variables for storing different items, such as states, to complex representations such as artificial neural networks, artificial immune systems, cognitive models etc.

B. Definitions of the terms:

Validation is the process by which we can determine if the model is a representation of the system.[3]. This is always performed while keeping the specific abstraction by the designer in mind. Verification is basically the debugging of the system where we ensure that the model that we build is working correctly. Credibility is achieved when the decision-makers and other key project personnel accept the model as well as its results as "correct".

C. Correlation with VOMAS?

VOMAS approach has been designed to cater for all kind of agent-based models. As such, it has capability to monitor spatial as well as non-spatial concepts in agent-based models.

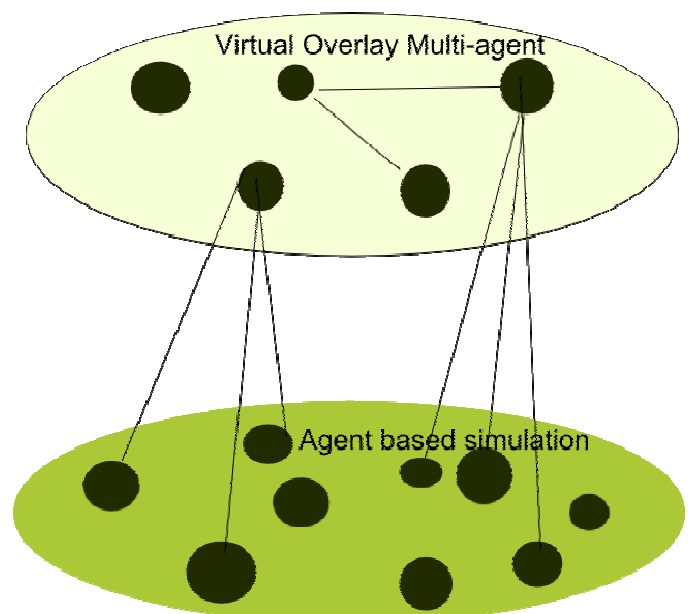


Fig. 1 VOMAS relation with an Agent Based Model

D. Verification & Validation of agent-based models

One sure way to establish the validity of agent-based model is to have Subject Matter Experts, who give the specification as well as examine the results and logs of simulation runs. VOMAS approach allows experts to be involved in the design of the agent-based model as well as the custom-built VOMAS from scratch. By involving SMEs from the start of the project, which are essentially equivalent to clients in the software engineering domain, VOMAS approach allows the simulation study to be a stronger candidate for success.

III. VALIDATION USING VOMAS

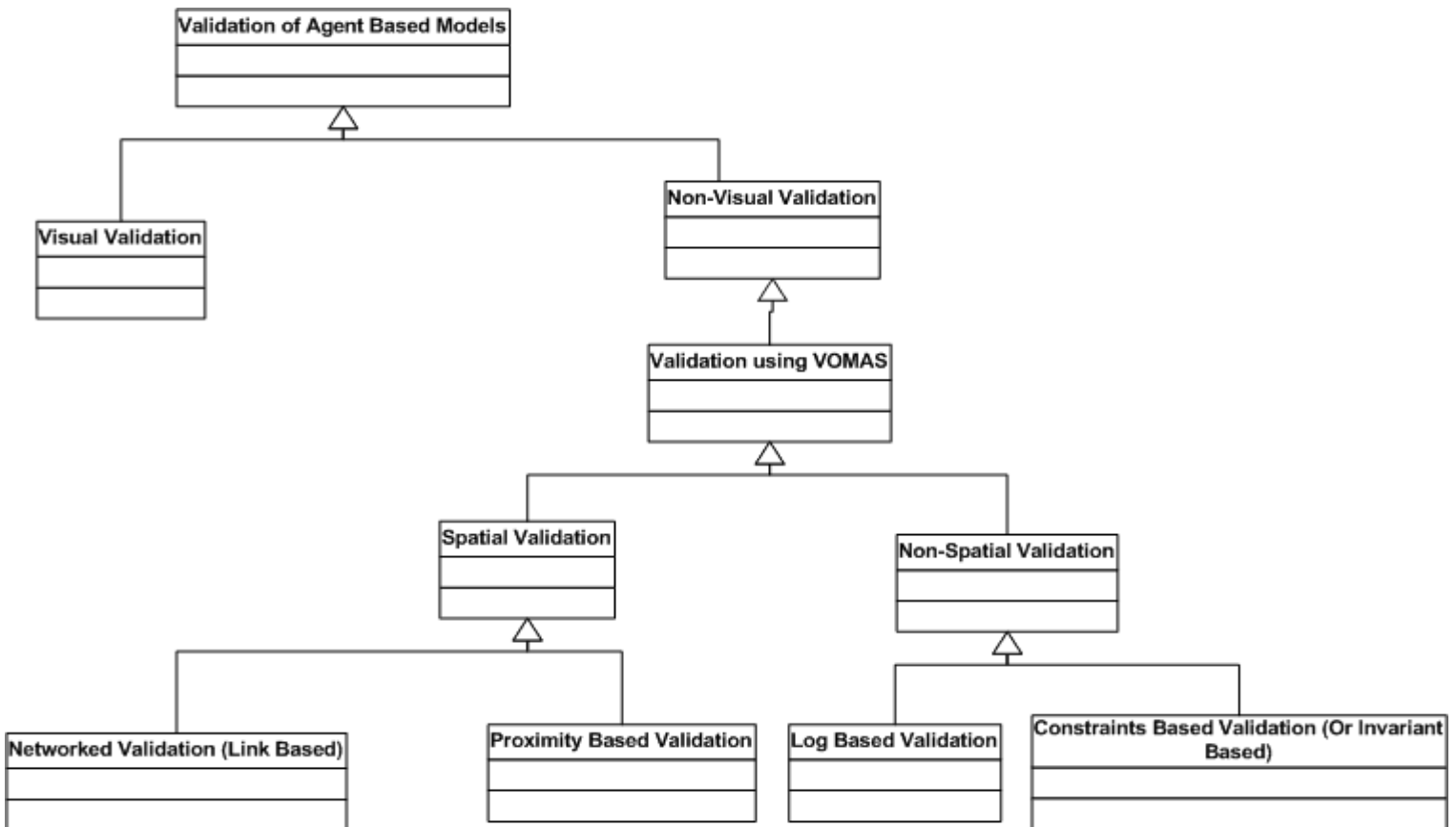
A. Validation in agent based simulations

To understand VOMAS, let us examine figure 1. The Virtual Overlay Multi-agent System is created for each simulation model separately by a discussion between the simulation specialist as well as the SMEs (Subject Matter Expert). When the actual simulation is executed, the VOMAS agents perform monitoring as well as logging tasks and can even validate constraints given by the system designer at design time.

B. A Taxonomy of Agent-Based Validation techniques using VOMAS

Now, let us examine how agent-based models are structured. Since agent based models have one or more agents, what these agents really mean in the real-world is entirely up to the designer of the simulation. These elements can be spatial in nature, where distance between agents in the simulation is important or else non-spatial, where there is no concept of distance in the simulation as shown in Fig. 2. In case of spatial models, it is also entirely possible that the exact distance may not be important, but the links between agents could be important. An example of this is HIV based models, where interaction between agents can be shown as links. A detailed description of each of these follows.

Fig. 2 A Taxonomy of Agent Based Validation techniques



1. Visual Validation:

Visual validation is a face validation technique based on an animation based validation technique where the

SME can examine the animation to see if the behavior appears to be similar to that expected in the actual domain.

2. Validation using VOMAS:

In case of VOMAS, we can validate both spatially as well as non-spatially.

3. Spatial Validation:

In spatial validation, the placement of agents in the simulation is important. This includes the placement of some of the VOMAS agents, which interact with the actual agent based simulation.

4. Non-Spatial Validation:

In non-spatial validation, the actual distance is not important. These could be used to validate for aggregate data and constraints/invariants etc.

5. Networked or Link-Based Validation

In spatial validation, it is possible that the actual placement is less important than the links between them. In case of social simulation, the example could be links to show social network friendships. In case of computer science based networks, these could represent e.g. Connectivity of Peer-to-Peer overlay networks.

6. Proximity Based Validation

In this case, the actual proximity of agents to each other and especially to VOMAS agents is important. An example of this is pred-prey models where

VOMAS agents can verify certain characteristics of agents passing by them at a certain time.

7. Log based validation:

In log based validation, the SME can specify what things to be watched and logged so that they can be examined after the fact and see how e.g. the populations evolved over time, or else how wireless sensor networks lost their power over time etc.

8. Constraint-based validation or Invariant Based Validation:

It is entirely possible that the SME says that there are certain constraints, which should never be violated in a certain simulation experiment. If these were ever to be violated, then the simulation system should notify the user via some console or else log the event as a

1) *Verify the Model*

The SME verifies the model by means of execution of the simulations by the Simulation Specialist. The detailed verification (debugging) is checked by the simulation specialist but in case of any ambiguity, the SME can be referred.

2) *Validate the Model*

This validation is done in three ways

a. Validation using animations:

This validation is face validation by the SME by means of analyzing the animations.

b. Validation using Logs

In this case, logs are generated based on watches specified by the SME. These logs

Fig. 3 Use case model of simulation model design and V&V



special case. E.g. Wolves must never all die in a wolf-sheep predation. If all of the wolves die, then the simulation needs to be stopped etc. as further data collection exercise might not be useful.

C. *Analysis of VOMAS*

The analysis of VOMAS has been conducted based on a scenario-modeling approach. In figure 3, we see the use cases, some of which are described below. The rest should be self-explanatory and we are not listing them for shortage of space:

show after the fact, the entire scenarios like black boxes from airplanes.

c. Validation using Invariants

These can be cases where the SME wants either immediate feedback even while running large scale parameter sweeps. So, if the invariants or constraints are ever violated, the user can be notified. Or at least, this is definitely logged in the simulation log.

3) Design and Develop Models

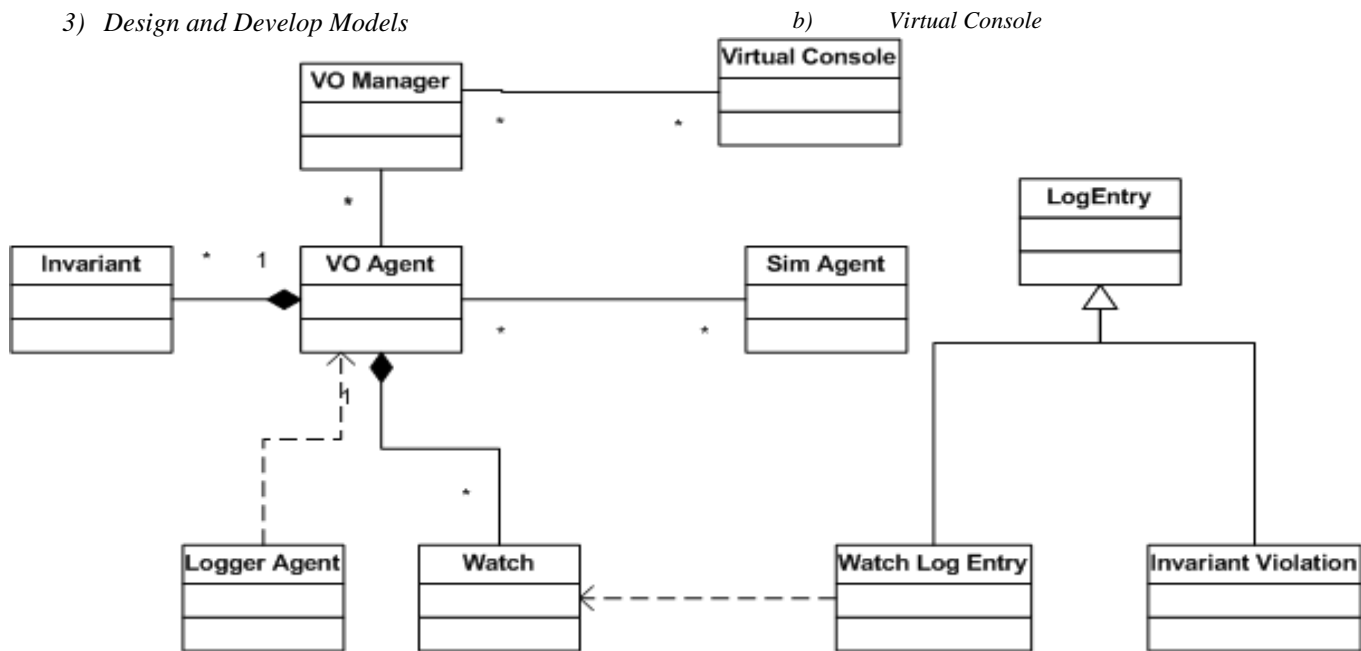


Fig. 4 Class diagram of agents in a VOMAS

This use case is to be conducted by the simulation specialist in conjunction with the SME.

D. Design of VOMAS

1) Motivation

One of the most popular approaches in Validation is the three step approach given in [28] . The approach has the following steps:

- a) Build a model that has high face validity.
- b) Validate model assumptions
- c) Compare the model input-out transformations to corresponding input-output transformations for the real system.

VOMAS has been designed to cater for both face validity as well as model assumptions and io-transformations. Model assumptions are ensured by the use of invariants. Face validation is ensured by means of various techniques based on spatial and non-spatial validation and animation-based validation. IO-transformations are ensured by means of essential logging components. Thus, in other words VOMAS provides the complete validation package.

2) Description of Class Diagram

In figure 4, we see the class diagram of the VOMAS agents and how they interact with the agents in the simulation. The description of each of these agents is given below:

a) VO Manager

VO manager agent is the key agent handling the interaction of all of the other agents.

b) Virtual Console

Virtual Console agent is an agent, which can be used to dynamically display various messages at run-time.

c) Invariant

Invariant is any condition, which the designer of the VOMAS and the agent-based simulation, feels that must not be violated during the execution of the simulation. If the Invariant is violated, the violation is logged.

d) Logger Agent

The logging capability is provided by the Logger Agent.

e) Watch

If the designer of the system wants some value to be observed, it can be made a watch.

f) Watch Log Entry

Each watch can also be logged as a logged entry.

g) Invariant Violation

Invariant violations can be logged at run-time to the Console Virtual agent or else the log as a log entry.

h) Log Entry

The base class of all log entries.

i) Sim Agent

This is an agent which is part of the agent based simulation model.

j) VO Agent

These are agents which can be located spatially or non-spatially to monitor the entire simulation.

IV. CASE STUDY

Here, we present application of a VOMAS to an agent-based simulation mode of the "Simulation of the research process". Recently an agent-based simulation model of researchers attempting to present research in International publication venues was presented in [5]. We demonstrate how to develop and use the associated VOMAS on this model.

A. The Publishing Researchers' model

In the publishing researcher model, the abstraction is that researchers are modeled as agents in the simulation. The higher the publications of an agent, the higher the agent goes. Thus space in this simulation model is essentially used to show the capability of the researcher. A screenshot of the simulation model is shown in fig 5. For more details, the interested reader is advised to consult the original article. The model has been developed using NetLogo [29]. So, let us formally define some of the entities involved:

SME: An Expert Researcher with experience of publishing in various venues.

Objective of Simulation Study: To examine how the policies of researchers in selection of publication venues impacts an overall organization.

Example Invariant:

Basis: In a particular simulation experiment, enough time of simulation run should be given to ensure that journal preferring researchers publish at least ten times during the simulation.

Invariant: If simulation stops before each journal preferring researcher is able to publish at least ten times, note an invariant violation in the console and/or the log.

Example watches:

Measure the total number of researchers with the best policy.
Measure the number of researchers above a certain threshold.
Measure the number of overall publications.

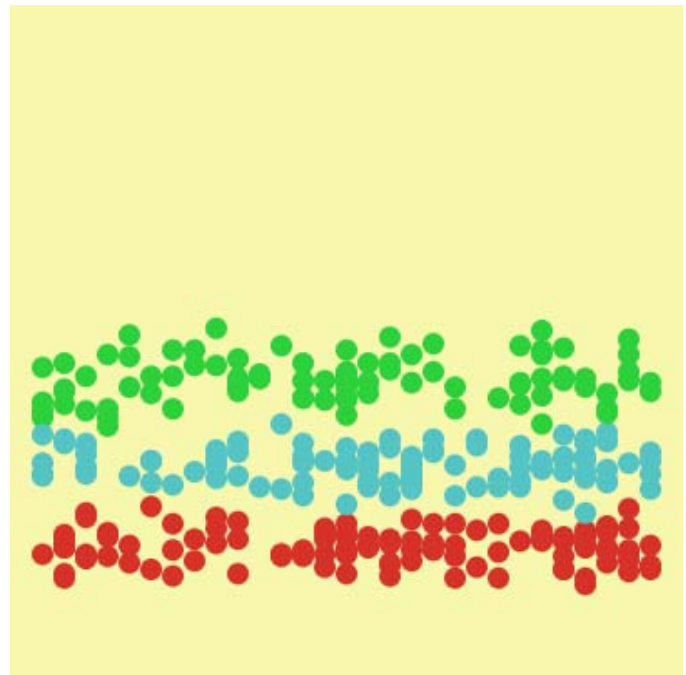


Fig. 5 Screenshot of the researchers' model [5] showing researchers according to their publication count. (Lime = Conference preferring, Red = Journal Preferring, Cyan = No Preference)

V. CONCLUSION AND FUTURE WORK

In this paper, we have presented a novel framework for the validation of agent based simulation models. We have given a description of how VOMAS agents can be constructed for validation. As a case study, we have shown its application on an existing published model. In the future, we shall apply VOMAS on various types of simulation models and demonstrate how it can be effective in validation. Some of the models we intend to explore VOMAS application on, include pred-prey models, tumor growth models, Peer-to-Peer unstructured overlay network models.

REFERENCES

- [1] O. Balci, "Verification, validation, and accreditation," in *Proceedings of the 30th conference on Winter simulation* Washington, D.C., United States: IEEE Computer Society Press, 1998.
- [2] J. Banks, J. S. C. II, B. L. Nelson, and D. M. Nicol, *Discrete-Event System Simulation*, Fourth ed.: Peason Education, 2005.
- [3] A. M. Law, "How to build valid and credible simulation models," in *Simulation Conference, 2008. WSC 2008. Winter, 2008*, pp. 39-47.
- [4] M. Niazi and A. Hussain, "Agent based Tools for Modeling and Simulation of Self-Organization in Peer-to-Peer, Ad-Hoc and other Complex Networks," *IEEE Communications Magazine*, vol. 47, No. 3, pp. 163 - 173., March 2009.
- [5] M. Niazi, A. Hussain, A. R. Baig, and S. Bhatti, "Simulation of the research process," in *Winter Simulation Conference*, Miami, FL, 2008, pp. 1326-1334.
- [6] M. Niazi, "Self-organized customized content delivery architecture for ambient assisted environments," in *Proceedings of the third international workshop on Use of P2P, grid and agents for the development of content networks*, Boston, MA, USA, 2008, pp. 45-54.

- [7] N. Gilbert and K. G. Troitzsch, *Simulation for the social Scientist*, Second ed.: McGraw Hill Education, 2005.
- [8] C. M. Macal and M. J. North, "Agent-based modeling and simulation: desktop ABMS," in *Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come* Washington D.C.: IEEE Press, 2007.
- [9] A. Siddiqi, M. Niazi, F. Mustafa, H. Bokhari, A. Hussain, Noreen Akram, S. Shaheen, F. Ahmed, and S. Iqbal, "A New Hybrid Agent-Based Modeling & Simulation Decision Support System for Breast Cancer Data Analysis " in *ICICT Karachi, Pakistan: IEEE Press*, 2009.
- [10] J. M. Galán, L. R. Izquierdo, S. S. Izquierdo, J. I. Santos, R. d. Olmo, A. López-Paredes, and B. Edmonds, "Errors and Artefacts in Agent-Based Modelling," *Journal of Artificial Societies and Social Simulation*, vol. 12, no. 11, 2009.
- [11] T. W. Lucas, S. M. Sanchez, F. Martinez, L. R. Sickinger, and J. W. Roginski, "Defense and homeland security applications of multi-agent simulations," in *Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come* Washington D.C.: IEEE Press, 2007.
- [12] A. Ilchinski, "Exploring self-organized emergence in an agent-based synthetic warfare lab," *Kybernetes*, vol. 32, pp. 38 - 76, 2003.
- [13] M. Chli and P. De Wilde, "The Emergence of Knowledge Exchange: An Agent-Based Model of a Software Market," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 38, pp. 1056-1067, 2008.
- [14] M. Cartier, "An Agent-Based Model of Innovation Emergence in Organizations: Renault and Ford Through the Lens of Evolutionism," *Comput. Math. Organ. Theory*, vol. 10, pp. 147-153, 2004.
- [15] C. Bianchi, P. Cirillo, M. Gallegati, and P. A. Vagliasindi, "Validating and Calibrating Agent-Based Models: A Case Study," *Comput. Econ.*, vol. 30, pp. 245-264, 2007.
- [16] C. Bianchi, P. Cirillo, M. Gallegati, and P. Vagliasindi, "Validating and Calibrating Agent-Based Models: A Case Study," *Computational Economics*.
- [17] J. H. Miller and S. E. Page, *Complex Adaptive Systems: An Introduction to Computational Models of Social Life*: Princeton University Press, 2007.
- [18] J. S. Hodges and J. A. Dewar, "Is It You or Your Model Talking? A Framework for Model Validation," RAND Corporation, Santa Monica, California.
- [19] G. Fagiolo, C. Birchenhall, and P. Windrum, "Empirical Validation in Agent-based Models: Introduction to the Special Issue " *Computational Economics*, vol. 30, pp. 189-194, October, 2007 2007.
- [20] S. Moss, "Alternative Approaches to the Empirical Validation of Agent-Based Models," *Journal of Artificial Societies and Social Simulation*, vol. 11, no. 15, 2008.
- [21] U. Wilensky and W. Rand, "Making Models Match: Replicating an Agent-Based Model," *Journal of Artificial Societies and Social Simulation*, vol. 10, p. 2, 10/31 2007.
- [22] A. Schmid, "What is the Truth of Simulation?," *Journal of Artificial Societies and Social Simulation*, vol. 8, p. 5, 10/31 2005.
- [23] O. Barreteau and e. al., "Our Companion Modelling Approach " *Journal of Artificial Societies and Social Simulation*, vol. 6, 2003.
- [24] M. Makowsky, "An Agent-Based Model of Mortality Shocks, Intergenerational Effects, and Urban Crime," *Journal of Artificial Societies and Social Simulation*, vol. 9, No. 2, 2006.
- [25] M. Cossentino, G. Fortino, A. Garro, S. Mascillaro, and W. Russo, "PASSIM; a simulation-based process for the development of multi-agent systems," *Int. J. Agent-Oriented Softw. Eng.*, vol. 2, pp. 132-170, 2008.
- [26] G. Fortino, A. Garro, and W. Russo, "From Modeling to Simulation of Multi-Agent Systems: An Integrated Approach and a Case Study," in *Multiagent System Technologies, LNAI 3187*: Springer-Verlag, 2004, pp. 213-227
- [27] R. S. Pressman, *Software Engineering, A Practitioner's Approach*, Sixth ed.: McGraw Hill, 2005.
- [28] T. H. Naylor and J. M. Finger, "Verification of Computer Simulation Models," *Management Science*, vol. 2, pp. B92-B101, 1967.
- [29] U. Wilensky, "NetLogo," Evanston, IL, 1999: Center for Connected Learning Comp.-Based Modeling, Northwestern University, 1999.

Agent-based modeling and simulation of multi-project scheduling

José Alberto Araúzo, Javier Pajares, Adolfo Lopez-Paredes
Social Systems Engineering Centre (INSISOC)
University of Valladolid
Valladolid (Spain)
{arauzo,pajares,adolfo}@insisoc.es

Juan Pavón
Facultad de Informática
Universidad Complutense de Madrid
Madrid (Spain)
jpavon@fdi.ucm.es

Abstract—There are no analytical solutions for the problem of dynamic scheduling of resources for multiple projects in real-time. Mathematical approaches, like integer programming or network based techniques, cannot describe complexity of real problems (multi-projects environments have many interrelated elements), and have difficulties to adapt the analysis to dynamics changes. However, this complex problem can be modeled as a multi-agent system, where agents negotiate resources through an auction inspired mechanism. Agents can be used to represent projects and resources. Projects demand resources for fulfilling their scheduled planned work, whereas resources offer their capabilities and workforce. An auction inspired mechanism is used to allocate resources to projects and the price of resources emerges and changes over time depending on supply and demand levels in each time slot. By means of this multi-agent system, it is possible to overcome most of the problems faced in multi-project scheduling such as changes in resources capabilities, allocation flexibility, changes in project strategic importance, etc.

Keywords—agent-based modelling; agent-based simulation; multi-project environments; auction based resources allocation; project scheduling.

I. INTRODUCTION

The problem of allocating resources for multiple concurrent projects appears in large cases of service and manufacturing organizations. A paradigmatic example can be an engineering projects office. This organization makes different kinds of projects that are proposed at any time, which must be handled in a given time frame. Each project consists of a number of activities (calculations, design, checks, budgeting, etc.) that are performed by workers and with some precedence relationships. The workers can perform one or several activities according to their skills. Decision makers have to reject inadvisable projects and decide which resources will be allocated to which projects and when.

Previous decisions have high impact in the office's profit. In order to achieve strategic goals it is important to give priority to projects, and to allocate activities to the most efficient workers at the appropriate time. Because of this, before executing projects it is advisable to make a schedule that optimizes the allocation of resources.

Classical methods are based on mathematical programming and can solve this problem when the complexity is low. And there are some heuristics and meta-heuristics that are able to provide good schedules for more complex problems [9]. The traditional scheduling and control systems propose hierarchical and centralized architectures, where a classical scheduler system that has a global model of the multi-project environment makes schedules according to the current state of the system. Hans et al. [4] review existing literature in hierarchical approaches and propose a generic project planning and control framework for helping management to choose between planning methods, depending on organisational issues.

But these techniques are not flexible or robust enough, and have difficulties to consider many real factors. In addition, real environments undergo frequent changes (new resources, new technologies) that force to modify the scheduling system. The traditional scheduling and control systems, which are based on hierarchical and centralized architectures, have not enough flexibility to adapt themselves to the dynamism and complexity of multi-project environments.

These issues have motivated, in last years, successive proposals are appearing to improve the scheduling and control in a multi-project environment. The paradigm of Multi-agent Systems (MAS) can help to find solutions, especially in cases where some social behaviour emerges. This paper shows an agent-based approach for online dynamic scheduling and control in multi-project environments that takes advantage of the ability of agents to negotiate and adapt to changing conditions. The MAS has basically two types of agents: projects managers and resources managers.

Projects have scheduled work to be done by different resources. Resources are endowed with some capabilities (knowledge, work force, etc.) that are needed to do the work. Projects demand resources over time and resources offer their capabilities and time availability. There is an auction process, and the price of resource-time slots emerges endogenously as a result of supply and demand. The design of the auction process uses a technique that has been proposed for distributed scheduling in the literature [8], [14], [11].

This agent-based approach has two distinctive aspects with respect to other works: the integration of strategic decisions

(accept or reject new projects) and operative aspects (resource allocation), and the ability to manage resource flexibility. This allows managers to study the advisability of increasing the flexibility of resources.

The next section introduces the role of agent-based modeling and simulation in project scheduling. Section 3 presents the MAS for the real-time scheduling problem, which has been specified with an agent-oriented modeling language, INGENIAS [10]. This has been the basis for implementing a simulation, which is described in section 4, and whose results are discussed in section 5. Finally, section 6 presents main conclusions of using this agent-based modeling and simulation approach.

II. AGENT ORIENTED MODELING AND SIMULATION FOR REAL-TIME SCHEDULING OF MULTIPLE PROJECTS

Multi-projects environments are complex and dynamic systems. They include many components and dependencies, and many changes may occur in the execution of projects. Moreover, projects are inherently distributed; each task may be completed by different resources or in different geographical locations and each project manager may be in different places.

MAS have been shown to deal with problems of *complexity*, *openness* (components of the system are not known in advance, can change over time, and are highly heterogeneous, dynamic in project management terms), with *dynamical and unknown environments changing over time* (uncertainty) and *ubiquity* (the activity is distributed over the complete structure) [5] [12].

In the particular case of multi-project systems, the agents can be abstracted as tasks, resources, project managers, etc. This design enables to distribute the management system in elemental components directly identifiable in the target system, and hence giving the opportunity to create systems easier to design, to adapt and to maintain. Moreover, since the system is distributed according to its structure, any change in the structure can be easily translated to the management system.

This decentralized approach facilitates the design of market mechanisms to solve the scheduling problem by means of distributed approximations [2]. Recently, Lee, Kumara and Chatterjee [7] have proposed an agent-based dynamic resource scheduling for multiple distributed projects using market mechanisms. Following the same research line, Confessore et al. propose in [3] another iterative combinatorial auction mechanism. Other examples of agent-based approaches in project management can be found in the works of Kim and colleagues [6], Wu and Kotak [13], and Cabac [1].

III. A MAS MODEL FOR MULTIPLE PROJECT SCHEDULING

The system can be modeled with two types of agents representing project and resource managers. Agents have the ability to interact with each other. In this case, it is important to define an auction protocol for project agents to compete for the use of resources. Resource Manager Agents interact with project agents to inform on the status, capabilities and cost at each specific time. A third type of agent is included in the

system to create new agents and monitoring the global behavior.

A. Project Manager Agents

Each project is associated to a *Project Manager Agent*. The system is considered dynamic: while some projects are being developed other projects can be included or rejected in real-time, which implies the creation and deletion of the corresponding agents.

At any instant t there are I projects in the system, each one denoted by i . Each one is characterized by a value V_i , that can be interpreted as the revenue obtained for the project, a weight w_i representing the strategic importance given to the specific project, a desirable delivery date D_i , a limit delivery date D_i^* , which cannot be exceeded, an arrival date of the project to the system, B_i , and a limit answer date R_i that represents the latest date to decide whether to accept or reject the project.

Each project i consists of J_i activities, each one denoted by ij , where $i \in \{1, 2, \dots, I\}$ and $j \in \{1, 2, \dots, J_i\}$. Every activity j of a project i is associated with a competence $h(i,j)$. Any activity ij with a given competence $h(i,j)$ can be performed by a resource m just if m is endowed with the competence $h(i,j)$. The duration of the activity ij depends on the resource assigned to perform it. The duration of activity ij in resource m is denoted as d_{ijm} . It is calculated according to $d_{ijm} = d_{ij} / e_{m,h(i,j)}$, where d_{ij} is the standard duration of activity j of project i and $e_{m,h(i,j)}$ is the efficiency of resource m to perform the competence $h(i,j)$.

This first simplified model assumes that the activities of any project should be performed sequentially in the order defined by j and only one resource can be assigned to an activity. There is also the assumption that once some resource has begun a task, the activity cannot be interrupted; the resource needs to finish it to be assigned to any other activity.

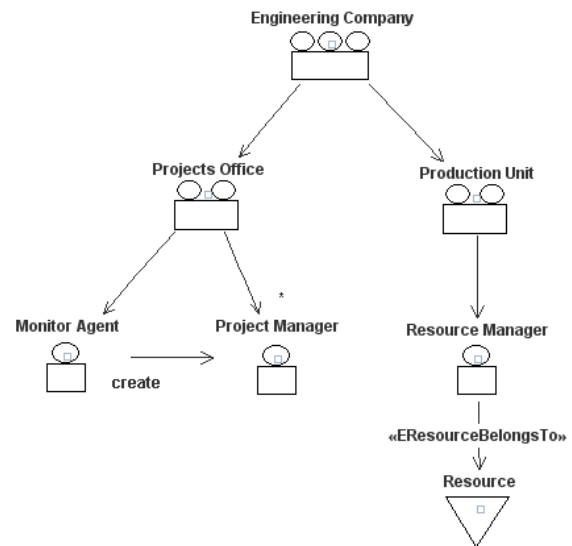


Figure 1. MAS organization model (with INGENIAS notation [10]). The diagram shows an organization (Engineering Company), which has several departments (Projects Office and Production Unit). The Projects Office has one Monitor Agent and several Project Manager Agents. The Production Unit has Resource Managers that take care of the use of Resources.

B. Resource Manager Agents

A resource is modelled as a *Resource Manager Agent*. There are M resources, which can be assigned simultaneously to one activity. Each resource is endowed with a given cost rate per unit of time, c_m ($m \in \{1, 2, 3 \dots M\}$), and a subset H_m of competences that can be performed ($H = \{h_1, h_2, \dots, h_K\}$ is the set of competences that are necessary to complete the projects).

Each resource has a certain grade or ability to perform a competence. Therefore, the work capacity of resources can be symbolized by means of a vector of abilities per resource $e_m = (e_{m1}, e_{m2}, \dots, e_{mk})$, where $e_{mf} \geq 0$ shows the ability degree of resource m to perform the competence h_f . If $e_{mf} = 0$ then the resource m has not the competence h_f , if $0 < e_{mf} < 1$ the resource is able to perform inefficiently the competence h_f , if $e_{mf} = 1$ it has standard efficiency to perform the competence, and if $e_{mf} > 1$ it will do it efficiently.

C. Monitoring Agent

A Monitoring Agent has the responsibility to visualize the current state of the system to the user. Moreover, this agent allows the user to create new *Project Manager Agents*, as shown in Figure 1.

IV. AGENT WORKFLOWS AND INTERACTIONS

The agent workflows and interactions must be designed in order to maximize the global efficiency of the system, which will be evaluated by the average benefit obtained in a certain time interval T according to:

$$Efficiency = \frac{B_T}{T} = \frac{\sum_i (V_i - Cost(i))}{T} \quad (1)$$

for all projects i that are finished in T , $Cost(i)$ is the cost to complete the project i . This cost has two components, the direct resource cost and the delay cost:

$$Cost(i) = \sum_j C_{\bar{m}(j)} \cdot \frac{d_{ij}}{e_{ij\bar{m}}} + w_i \cdot (D_i - F_i)^2 \quad (2)$$

The first addend corresponds to the direct resource cost to finish each activity j . $\bar{m}(j)$ denotes the resource selected to comply with activity j . The second addend is the delay cost associated with the project, where F_i is the real delivery date.

The problem considers the decision to reject projects. This could happen in any of the following cases:

- The revenue obtained from the project does not compensate the costs.
- The scheduling exceeds the D_i^* of the project.
- The impact on the scheduling of the rest of the projects is not acceptable. This may happen for two causes. First, if the new project obliges to delay a committed

project beyond D_i^* , it will be rejected. If not, but the inclusion of the new project increases the delay costs of the other projects more than the direct benefit obtained for the project, it will also be rejected.

A. Auction Interactions

At any time, the system has as many Project Manager Agents as projects are ordered. Each one represents a particular project characterized by its tasks, precedence relationships, due date, value, local programs and their execution state. Their goal is to look for contracts with resources that can perform the required activities and hence completing successfully the project. In order to achieve their goal, Project Manager Agents make plans that take into account only their own activities (local schedule).

The decision-making process is decentralized as it emerges from interactions among the agents in an auction process. Each project manager creates its own schedule (local schedule) by taking into account its own project goals and its own knowledge. This procedure can bring incompatible local schedules (several projects try to use the same resource at the same moment). Moreover, the local schedules can be globally inefficient (profitable projects are rejected; most important projects have delays; etc). These difficulties that arise from the autonomy of each agent are solved with a market mechanism that ensures that local schedules are nearly compatible and globally efficient according to the expression (1). This auction based multi-project scheduling approach is founded on Lagrangian Relaxation [8][11][14], a decomposition technique for mathematical programming problems.

In order to apply the market metaphor, the periods when resources are available are subdivided in a set of small time intervals or time slots. Each time slot on each resource is modelled as a *good* that can be sold in an auction, where each resource acts as a seller. Thus, a local schedule will be a bundle of time slots that has been allocated to a project.

The number of sellers is equal to the number of resources in the system. Each resource proposes a price for the time slots from the current time to the end of the scheduling horizon. The scheduling horizon changes dynamically by coinciding with the latest time slot that some project has asked at any moment.

Each project agent plays the role of a *bidder* that participates in auctions by asking the Resource Manager Agents for the set time slots that it requires to execute its pending tasks at the current time. It will try to find a set of time slots (Z_i) through the resource pool while incurring the minimum possible local cost (LC_i). This cost has two components, the sum of the price of the selected time slots and the delay cost (expression 3):

$$LC_i = \sum_{m \in Z_i} p_{mt} + w_i \cdot (D_i - F_i)^2 \quad (3)$$

where p_{mt} is the price of the time slot (t) of the resource (m).

To select the set of time slots (Z_i) that minimizes their local cost, Project Manager Agents use a dynamical programming algorithm where all possible combinations of time slots and resources are considered [13]. In their decision, they take into account that only those resources endowed with the necessary competences can carry out a certain activity. Moreover, the number of time slots necessary to complete a task (duration) are determined according to the ability degree of the resource in the competence. Each project agent will regard as scheduling horizon the time slot that goes from the current time to the limit delivery date (D_i^*). If some project agent cannot find a set of time slots in such a manner that it allows to schedule tasks before D_i^* , with a smaller cost than its value (V_i), then it will not ask for any set of time slots. This implies that the project is unprofitable at the correspondent round of bidding and must be rejected.

Each Resource Manager Agent determines the price charged for the time slots with the purpose of reducing resource conflicts and maximizing their revenue. In order to get this goal a subgradient optimization algorithm is used to adjust prices at each round of bidding. By means of this algorithm the Resource Manager Agents increase the price of the time slots where there is conflict (more than one project manager has asked for this time slot) and reduce the price of the time slots that have not been demanded. The process of price adjustment and bid calculation continues indefinitely. At each round of bidding the resource conflicts will be reduced.

At the first round of bidding, the time slots prices for the resource (m) are equal to the resource cost rate (c_m). At the rest of bidding round, the prices will be updated by means of the expression 4. α^n is calculated according to [8].

$$p_{mt}^{n+1} = \max\{c_m, p_{mt}^n + \alpha^n \cdot g_{mt}^n\} \quad (4)$$

Where:

- p_{mt}^{n+1} is the price of the time slot (t) of resource (m) at the round (n+1)
- p_{mt}^n is the price of the time slot (t) of resource (m) at the round (n)
- α^n is the step at the round (n). It decreases when (n) increases.
- And ($g_{mt}^n = a_{mt}^n - 1$) is the subgradient, where a_{mt}^n is the demand of slot (t) of resource (m)

B. Contract Interactions

By means of the auction mechanism described above, project agents build compatible and globally efficient local schedules for their pending activities. Moreover, at the same time, agents interact through a complementary process to make firm agreements based on the local schedules that have been created by means of the auction process. These agreements determine fixed programs for earliest scheduled tasks. When

these agreements are obtained, project agents will never consider the tasks included as firm contracts as pending.

The global efficiency and the compatibility of local schedules depend on the degree of convergence of market prices to the equilibrium prices. If the prices get closer to the equilibrium price, they will be representative of the system state; they will have information about any system feature and local schedules will be compatible and globally efficient. If agents are making firm contracts when prices are not representative of the system state, then incompatibilities could take part. In these cases, the agents resolve incompatibilities by means of local schedule based heuristics rules. More exactly, when several activities use the same resource at the same moment, the activity that has been earliest programmed in local schedule will have priority to be contracted in firm agreements. Although this heuristic does not ensure global efficiency, it will achieve perfect compatibility in final decisions.

V. SIMULATION AND RESULTS

The system has been implemented and simulated with different scenarios. Here the analysis focuses on the role of resource capabilities and the option of project rejection. The first scenario shows a simple case to illustrate the main features of the system, in the next subsection. This is followed by a dynamic scenario in order to evaluate the system performance in evolving complex environments.

A. Simple Case Study

Consider three different resources (R1, R2 and R3), endowed with the competences C1, C2 and C3 respectively. TABLE I. shows a portfolio of five projects, and the tasks needed to complete each project. Each task is defined by means of the pertaining competence and expected standard time to be completed.

TABLE I. SIMPLE CASE STUDY

Proj.	Tasks			Arrival date	Starting Date	DD1	DD2	Value
	Task 1	Task 2	Task 3					
P1	C1 50	C2 25	C3 30	0	0	120	180	10000
P2	C3 40	C1 45	C3 10	0	0	180	240	12000
P3	C2 35	C1 40	C2 25	0	0	120	180	30000
P4	C3 30	C1 50	C2 10	50	90	150	270	15000
P5	C1 45	C3 20	C1 50	50	90	150	270	30000

The arrival date is the date when the project is included in the system. Projects can start-up in the starting date; otherwise, they should have been rejected before this date. Due Date 1 (DD1) is the most desirable duration whereas Due Date 2 (DD2) is the maximum allowed. All the projects have a weight of 1.

Figures 2 and 3 show the system state at a given time (current time). In the upper area of the figures the relative duality gap evolution is presented. The prices of time slots are the solution of the dual problem and the duality gap is a

measure of the difference between the primal and dual objective function, so it quantifies the quality of the solution [8]. The relative duality gap is calculated as the duality gap divided by the dual solution. A small relative duality gap means that the prices are representative of the system state, thus, a good solution is achieved. The lower part of the figures present charts of resources. These charts show the tasks that each resource has performed until the current time (lower area of the resource charts) and the time slot prices (upper area of the resource chart). The time slots prices previous to current time are the prices when agents were doing firm agreements for those time slots. The prices later than current time are the estimated prices in the current round of the auction.

Figure 2. shows the system state at the moment 45, just before projects P4 and P5 arrive at the system. When the first projects (P1, P2 and P3) are included in the system, the duality gap is high, indication of a bad solution. But then, the price formation mechanism makes the prices to stabilise, and the gap becomes smaller. This means that prices are close to equilibrium.

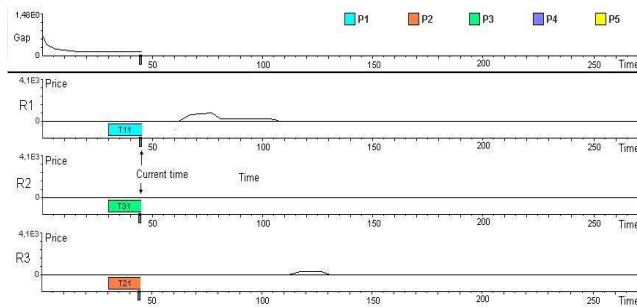


Figure 2. System state at the moment 45

Figure 3. shows the evolution of the tasks performed by each resource and the prices of the time slots after finishing the simulation. This figure shows how the duality gap increases when the projects P4 and P5 arrive to the system. At this moment previous prices did not reflect the new system state (new projects are in the system). After some time, the prices change to adapt themselves to the new system conditions, and the gap decreases again. It can be observed that the new prices are very different from previous prices. This happens especially in resource R1 where prices are very high.

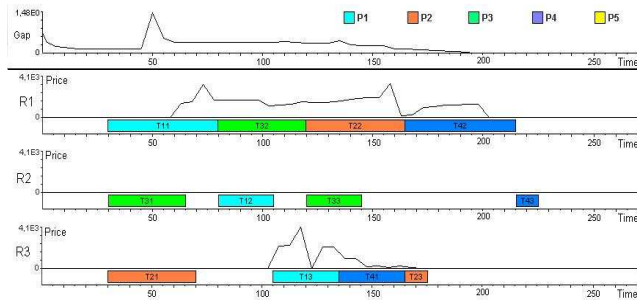


Figure 3. Tasks performed by resources. T_{ij} denotes Task j of project P_i .

Note that project P5 has been rejected although it has a high value, because its value was not available at time 0, when projects P1, P2, P3 were waiting to start-up. The calculus of the payment that projects have done for time slots (TABLE II.) shows that the same projects do a payment higher than their values. When projects P4 and P5 arrive at the system the prices of time slots of the resource R1 grow because P5 is able to pay higher prices to be performed. Although P5 accepts higher prices than other projects, P1, P2 and P3 cannot be rejected and finally they must pay the market prices. The final total value (B_T =total values of performed projects minus total delay cost) is 55700.

TABLE II. PAYMENT FOR TIME SLOTS PER PROJECT

Project	Value	Total payment
P1	10000	12719
P2	12000	11414
P3	30000	8298
P4	15000	6887
P5	0	0

The simulation not only gives the dynamic schedule and the refused projects, but the value of each resource as well. For instance, in Figure 3. the prices of resource R1 are very high during all time slots. This means that the resource competence is very valuable (bottleneck), so if the firm is going to be engaged in similar projects in the nearby future, it would be useful to include more resources with the same competences. On the other hand, prices of resources R2 and R3 are small, although they are working on different tasks during the simulation.

So, the possibility of enhancing the range of capabilities of resources R2 and R3 should be considered; for instance, in the case of human resources, this can be done by means of training.

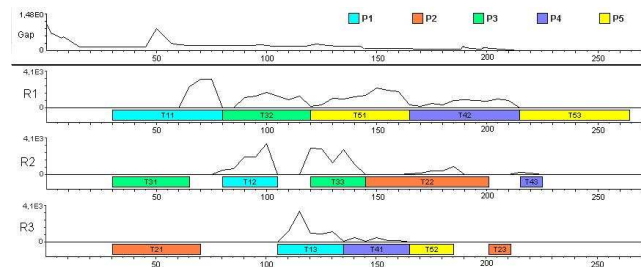


Figure 4. Tasks performed by resources (competences of resource R2 increased).

Figure 4. shows the evolution of the system when the resource R2 is also endowed with the competence C1 with efficiency 0.8. Compared with the previous case, now the price range is lower for resource R1 and higher for R2. Although the duration of task T22 of project P2 is smaller in resource R1 than in R2, now the system have to reallocate in real-time this activity to R2. So, R1 can perform in time the task T51 of the project P5. In this experiment, the project P5 is accepted and executed, and the total value has been increased from 55700 to

69369. This shows that the system is capable to use the flexibility of resource R2 to improve in real-time the global performance.

B. Complex Dynamic Scenario

In order to check the system performance in very dynamic environments, consider 12 projects (table 3) that arrive at the system every 20 units of time (first P1, second P2, ..., and finally P12). In TABLE III. DD1 and DD2 are relating to starting date. Resources and competences are similar to the previous case study.

TABLE III. DYNAMIC PORTFOLIO OF PROJECTS (COMPLEX SCENARIO)

Proj.	Tasks			DD1	DD2	Value
	Task 1	Task 2	Task 3			
P1	C1 50	C2 25	C3 30	60	150	10000
P2	C3 40	C1 45	C3 10	60	120	15000
P3	C2 35	C1 40	C2 25	60	120	6000
P4	C3 30	C1 50	C2 10	90	120	7000
P5	C1 45	C3 20	C1 50	60	120	8000
P6	C3 10	C2 45	C1 20	120	150	7000
P7	C1 20	C2 10	C3 30	60	150	15000
P8	C3 40	C1 45	C3 50	90	120	10000
P9	C2 35	C1 10	C2 45	60	120	15000
P10	C3 30	C1 15	C2 10	60	120	10000
P11	C1 15	C3 50	C1 10	90	150	7000
P12	C3 35	C2 50	C1 20	60	120	12000

We have done several simulations by changing two types of parameters: the response period and the set of competences of resources. The response period is the time interval between the arrival date and the starting date. During this period, projects wait in the system for rejection or acceptance decision. If this period is long, more projects are waiting for decision simultaneously, so decisions will be more efficient.

We have simulated three competence distribution cases: case A (R1 has the competence C1, R2 the C2, R3 the C3), case B (R1 C1, R2 C1 and C2, R3 C3), and case C (R1 C1, R2 C1 and C2, R3 C2 and C3).

Figure 5. shows the total values obtained in different experiments. Each curve represents the value variation for cases (A, B and C) when the response period increases. Note that the system efficiency is higher when the response period increases and when the resources are more flexible (they have more competences). This shows that in this scenario the system performance is suitable; the software is able to manage complexity to improve the global efficiency.

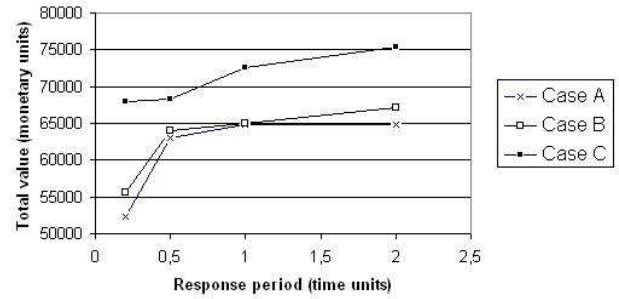


Figure 5. Total value in complex experiments

VI. CONCLUSIONS

Although project management literature has been mainly concerned with managing individual projects, in practice firms usually work in dynamic and complex multi-project environments.

We propose a multi-agent system and an auction mechanism for online dynamic scheduling in multi-project environments. Projects have tasks to be completed, so they compete for the resources endowed with the capabilities required to do some pieces of work. The prices of resources emerge endogenously by means of an auction process.

We show some of the possibilities of this multi-agent approach to deal with some of the decisions that managers need to take within multi-project environments. The system allocates dynamically resources to projects, and decides what projects to accept or reject taking into account project value, profitability and (feedback) operational information. We also show how it is possible to discover which resources are the most valuable, so they should be added to the firm.

This approach contributes to fill the gap between the literature in portfolio project management (usually focused on corporate strategy and finance) with the work in multi-project management (mainly concerned with operational issues, scheduling and resource allocation).

ACKNOWLEDGMENTS

This work has been done in the context of the following projects: (1) "Agent-based Modelling and Simulation of Complex Social Systems (SiCoSSys)", supported by Spanish Council for Science and Innovation, with grants TIN2008-06464-C03-01 and TIN2008-06464-C03-02; (2) "ABACO VA006A09", (3) the Programa de Creación y Consolidación de Grupos de Investigación UCM-BSCH GR58-08, and (4) GR251/09 supported by the "Junta de Castilla y Leon".

REFERENCES

- [1] Cabac, L. "Multi-agent system: A guiding metaphor for the organization of software development projects". In: Carbonell, J.G. & Siekman, J., Multiagent System Technologies. Lecture Notes in Artificial Intelligence 4687. Springer: Berlin / Heidelberg, pp 1-12, 2007
- [2] Clearwater, S. "Market-Based Control: A Paradigm for Distributed Resource Allocation". World Scientific, 1996.
- [3] Confessore, G., Giordani, S. & Rismondo, S. "A market-based multi-agent system model for decentralized multi-project scheduling". Annals of Operations Research, 150, pp 115-135, 2007
- [4] Hans EW, Herroelen W, Leus R and Wullink G. "A hierarchical approach to multi-project planning under uncertainty". Omega 35, pp :563-577, 2007.
- [5] Jennings, N.R. & Wooldridge, M.J. Applying agent technology. Applied Artificial Intelligence, 9, pp 357-369, 1995.
- [6] Kim, K. & Paulson, J. "Multi-agent distributed coordination of project schedule changes". Computer-Aided Civil and Infrastructure Engineering, 18, pp 412-425, 2003.
- [7] Lee, Y.H., Kumara, S.R.T. & Chatterjee, K. "Multiagent based dynamic resource scheduling for distributed multiple projects using a market mechanism". Journal of Intelligent Manufacturing, 14, pp 471-484, 2003
- [8] Luh, P.B. & D. J. Hoitomt. "Scheduling of Manufacturing Systems Using the Lagrangian Relaxation Technique". IFAC Work Shop on Discrete Event System Theory and Applications in Manufacturing and Social Phenomena, Shenyang, China, 1991.
- [9] Newman K, Schwindt C., Zemmermann J. Book review of Project Scheduling with Time Windows and Scarce Resources: Temporal and Resource-Constrained Project Scheduling with Regular and Non-regular Objective Functions. 2nd Edition, Springer-Verlag, 2003
- [10] Pavón, J., Gómez-Sanz, J.: "Agent Oriented Software Engineering with INGENIAS". In: Marik, V., Müller, J., Pechoucek, M. (eds). Multi-Agent Systems and Applications III, 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003. Lecture Notes in Artificial Intelligence 2691. Springer-Verlag, Berlin Heidelberg, pp 394-403, 2003.
- [11] Wang, J., Luh, P.B., Zhao, X. and Jinlin Wang. "An Optimization-Based Algorithm for Job Shop Scheduling". Sadhana, a Journal of Indian Academy of Sciences, Vol. 22, Part 2, pp. 241-256, 1997
- [12] Wooldridge, M.J. An Introduction to Multiagent Systems. John Wiley & Sons Ltd: New York, 2002.
- [13] Wu, S. & Kotak, D. "Agent-based collaborative project management system for distributed manufacturing". Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, pp 1223-1228, 2003
- [14] Zhao, P. B. Luh, J Wang. "Surrogate Gradient Algorithm for Lagrangian Relaxation". Journal of Optimization Theory and Applications, Vol. 100, N° 3, pp 699-712, 1999

Quick Prototyping and Simulation with the INGENIAS Agent Framework

Jorge J. Gomez-Sanz
Facultad de Informática
Universidad Complutense
Madrid, Spain
Email: jjgomez@sip.ucm.es

Carlos Rodríguez Fernández
Facultad de Informática
Universidad Complutense
Madrid, Spain
Email: carlosrodriguez@computer.org

Juan Pavón
Facultad de Informática
Universidad Complutense
Madrid, Spain
Email: jpavon@fdi.ucm.es

Abstract—A major nightmare of software developers is having clients claiming the delivered software is not what they expected. Developers make an extensive use of prototypes to prevent such situations to occur. However, the role of simulations have not been studied enough. If an agent based simulation is intended, not all existing agent oriented methodologies are capable of it. It requires code generation capabilities, round-trip features, and components with predefined behaviors. This work introduces the ability of the INGENIAS Agent Framework to simulate MAS specifications and how this can be useful for agile development of agent-based applications.

I. INTRODUCTION

A software development needs to verify the system under construction satisfies a client needs. Requirements engineering has contributed strongly to the solution by guaranteeing the requirements are properly captured and processed. Software development processes have been modified to include the ideas of incremental development (segmenting the system functionality into coherent pieces and ordering them so that their sequential/parallel realisation leads to the final system), iterations (miniprojects focusing on the development of a concrete piece of functionality of the application). Expected products from this development process include several types of documents aiming to communicate with the client and tell them what is being done; and acceptance tests, to ensure the behavior of the application is the one expected. Despite these avances, prototyping seems to be a recurrent option.

Prototypes [1] have been extensively used as a effective mean of showing results to the client and for experimenting with part of the functionality the client demanded. However, the development of a prototype is incomplete without a proper environment that resembles the platform where the software will be deployed. Hence, essaying with different software architectures or showing a client how the software is expected to behave is expensive.

In this scenario, agent technology has been recognised as an affordable mean for producing prototypes and define simulations [2]. Nevertheless, actual prototyping and simulators are built ad-hoc. The facilities needed to facilitate rapid application development where a simulation is required are not present in most agent oriented methodologies. Remarkable exceptions are ADELFE [3] and PASSIM [4], which integrate

simulations into their development cycles. However, as discussed in section VIII, more effective code generation and specification-code synchronization facilities are needed.

In this aspect, INGENIAS [5] can improve existing proposals. INGENIAS is an agent oriented software engineering methodology which follows the model driven development paradigm [6]. As a result, it considers the MAS specification as the main product of the development and provides tools to transform this specification into executable code. INGENIAS takes advantage of the INGENIAS Development Kit for producing fully functional systems [7]. Within this kit, there is a distinguished module, the INGENIAS Agent Framework which determines how to interpret the MAS specification using as target platform JADE. This module is complemented with the Code Uploader and the AppLinker modules that provide round-trip engineering features, i.e., they upload changes made in the code back into the specification.

This paper discusses how INGENIAS can make use of the simulation concept in order to develop a software system. The discussion bases in the features provided by the INGENIAS Agent Framework, which is introduced in section II. The way this software can be used to simulate is introduced in section III. The introduction of the case study concerns to section IV. The INGENIAS solution to the case study is made in section V. The simulation made in this case study itself is introduced in section VI and evaluated in section VII. The related works are presented in section VIII. Finally, section IX introduces the conclusions.

II. THE INGENIAS AGENT FRAMEWORK

IAF stands for the INGENIAS Agent Framework. It is a framework developed along several years that enables a full model driven development. This means that a developer can focus most of its effort in specifying the system, converting a great deal of the implementation in a matter of transforming automatically the specification into code. This IAF permits to combine the classic approach for coding applications with modern techniques of automatic code generation. The resulting system is almost fully operational, reducing the amount of work of the developer in an relevant degree. Each produced MAS works over the JADE platform. Hence, additional tools existing for this framework can be applied as well.

A MAS in the IAF is constructed over the JADE platform. The MAS can be distributed along one or several containers in one or many computers. To enable this feature, the IAF has means of declaring different deployment configurations.

The running MAS will be connected to several non-agent applications providing the basic services. Hence, if the MAS has to interact with a user, there will be GUIs producing events according to user actions, and defining actuators for agents. These GUIs will be specified as *applications* at the specification level.

An important feature of the IAF is the relevance of interactions, which are considered first class citizens during specification and coding. An *interaction* in runtime is called a *conversation*. The interactions according to the IAF have the main purpose of transferring information from one agent to another. This information transfer is ruled by timeouts and initiation/collaboration conditions. Also, interactions can be aborted due to failures in the communication or simply because an agent did not answer within the timeout. Finally, the software code realising interactions consider cases where there may be several actors of the same time, i.e., supports the deliver of information to several recipients and the reception of the answer from several agents at the task level.

Tasks are important as well. An agent chooses to schedule a task for execution because the agent wants to attain a pursued goal. The tasks influence in the mental state by removing/adding information, starting conversations with other agents, or modify already existing conversations. Tasks support cardinality attributes associated to the inputs, so a task can use as input all instances of a certain information type or just a few.

Custom deployments permit the developer to define which types of agents will be used and what individual mental state will have during the start-up. This feature allows the developer to define different configurations of the system so that the developer can observe the behavior of the MAS under such conditions.

Testing is a recent addition to the IAF. A developer can define at the specification level what tests will be performed and to what configuration of MAS will be applied. The detailed definition of the test has to be handcrafted, though there are some software libraries that make this work easier.

III. BUILDING A SIMULATION WITH THE INGENIAS AGENT FRAMEWORK

In general, there are two approaches for simulation using the IAF: simulating the environment or simulating the environment and the application. In the first, MAS infrastructure is provided in order to represent different elements of the target runtime environment. Therefore, there is external software (the one recently developed and whose behavior is to be verified) and the environment built with a MAS. The external software would be docked to the MAS based environment by means of *application* entities. In the second, not only the elements of the application environment are provided, but parts of

the system-to-be are included as well, reusing directly predefined behaviors from the IAF.

A. Simulating the internals of the application

The IAF provides built-in predefined behaviors which can be used to simulate parts of the application. They are introduced following:

- **Task Raw simulation.** The developer defines no specific code for tasks, assuming the default behavior of tasks in the IAF. This behavior consists in consuming/reading the information declared as input and producing all the outputs declared in the specification. All produced instances of information entities are empty. This means that if an information entity has attributes, its instances will see these attributes exist, though they have empty values. Also, when a task creates an instance of an interaction, i.e., a conversation, the collaborators will be chosen randomly among existing valid ones (a valid agent type or a valid agent role according to the interaction definition). If one collaborator is defined as having cardinality greater than one, then multiple agents satisfying the requirements from the concrete interaction specification are chosen and incorporated automatically.
- **Application Raw simulation.** The applications are wrappers for non-agent software. The default code produced for such instances of applications does nothing. Nevertheless, if the specification declared a certain event has to be produced from the application, then the IAF generates a GUI from which the user can trigger the generation of an instance of the expected event. This new instance would be incorporated automatically into the mental state of the agent owning the application. If the specification declares an application type is owned by certain agent type, then an instance of an application can be accessible only to an instance of the agent type or to multiple instances of the agent type (this is implemented by means of a singleton pattern [8]). This is useful to represent shared resources and communication through the environment. In the later case, one agent performs an action over the application that triggers an event which is passed to all agents owning the application.
- **Interaction Raw simulation.** For each agent capable of initiating a conversation, the IAF generates a basic GUI that can trigger this conversation without having a task launching it. The conversation may not progress if the corresponding information that should be delivered/received does not exist. Therefore, a simulation of the interactions must be accompanied with a specialized deployment where the initial mental state of involved agents satisfies the requirements of the interaction at the specification level.

Using these pre-defined behaviors, the developer models the internals of the application using an agent oriented methodology, like INGENIAS, and proceeds to observe how the resulting MAS behave.

The simulation can get closer to the actual intended software by customizing more the behavior of the elements:

- Adding custom code to the tasks in the MAS. Instead of the default behavior (consuming input entities and creating new ones in the output), a developer can code more concrete behavior, like using existing APIs from applications to perform actions or detail which collaborators a conversation will have. This new code is inserted into a code component entity and used to replace the code in the generated task. These changes are maintainable through the use of the code uploader module of the INGENIAS Development Kit, which migrates changes made to tasks into existing code components in the specification.
- Adding initialization/shutdown code for applications. This code permits to properly construct/shutdown the application knowing during creation/shutdown time which agent will be assigned to it. This way, an application acquires a reference to its agent or agents.
- Modifying the API of the application code and providing a body to the methods. An application can be coded ad-hoc for a concrete development or act as mediator [8] to some external software (e.g. a database or some existing GUI). The API is synchronized with the specification by means of the AppLinker module of the IDK. This module analyzes the generated code of already generated applications looking for changes in the API with respect to the API stored in the specification. Differences are merged automatically so that the API in the specification is the same. This way, a developer can either modify the specification and let the system regenerate the code, or modify the API in the code and upload the new methods to the specification.

The degree of customization is a decision of the developer. The resulting MAS can become the intended system or remain as a proof that the MAS specification is valid for the problem. In the first, case, the specification and the customization of the generated code would progress towards the final system. In the second case, the developers would decide to realize the specification into a different agent platform or just reuse the acquired knowledge to be used within another methodology (agent oriented or not).

The IAF recognises automatically generated code, manually maintained code, and a hybrid mixture of both. All of them exist into separated folders. Hence, a developer can work safely in the *src* folder, creating classes regularly; delete safely the content of *gensrc* folder knowing that it can be completely generated from the specification; or customize the content of *permsrc* folder knowing that changes made will not be overwritten by successive code generation requests.

B. Simulating the environment

Once key elements to be simulated are identified, a developer can represent them as agents or as applications. It is an agent when its behavior of the simulated entity can be captured with goals and tasks and the execution of the corresponding tasks corresponds to the achievement of goals.

It is an application in other case. Being an application means an API is offered and that this API can be used within the tasks of the agent. Optionally, the application is expected to produce events in order to notify agents of changes. As explained in the previous section, these events can be asserted within a single or multiple owners.

The external software (software which already exists before the current MAS is developed) is wrapped into applications. Generated code for applications will act as a mediator [8] between the generated MAS and the external software. The initialization code for applications will be used to connect the mediator with the external software, while the shutdown code will do the opposite. These applications should offer the same API the external software does. If there are multiple APIs, a developer can choose to merge all of them into a single application or creating multiple applications holding each one separately.

Most likely, there will be agents representing the different types of users in the system, which could be humans or not. A developer will define a certain role capturing the generic behavior expected from that user. This behavior is supposed to be assumed by an agent or specialised by another role. Assuming there is a role with a task X having as input an entity type Y, the specialization can happen as follows. First, an agent can define a task XX having as input the entity Y. This will cause tasks XX will be executed instead of tasks X. Second, a new role extending the original role can be defined. This new role, can define a task XXX having as input the entity Y. This will cause tasks XXX are executed before the task X. In both cases, when task XX or XXX is executed, they, probably, will remove entity Y from the mental state, aborting this way any possible execution of previously scheduled X tasks.

With these two ways of capturing behaviors, a developer can create populations of users with varying behaviors. The resulting agents will perform actions over the existing applications, which are supposed to be connected to the software system currently under execution.

C. Analysing a simulation run

Once the MAS is defined, it is time to perform different runs of the system. The user can inspect visually the results by means of the IAF default GUI. Nevertheless, it is convenient to make use of a more exhaustive and objective analysis by means of studying the system logs. The INGENIAS Agent Framework produces logs with the produced events in a system run, so that they can be inspected and accounted later on. Registered events are:

- A task has been scheduled. The id and type of the task, as well as the id of the agent, are provided
- A new piece of information is added/removed to/from the agent mental state. The id and type of the entity as well as the id of the agent are provided.
- A task has been executed. The id and type of the task, as well as the id of the agent, are provided

- A task has been aborted. The id and type of the task, the expected inputs that were missing, as well as the id of the agent, are provided
- A conversation has been started. The id of the conversation, the interaction type, collaborators, and the id of the launcher agent, are provided
- An agent decides to participate into an requested conversation. The id of the conversation, the interaction type, collaborators, and the id of the launcher agent, are provided
- A message has been delivered. The id of the message, its content, sender and receivers are provided

Each event is marked with a timestamp (24 hour format and milliseconds format) obtained from the same hardware clock. Therefore, this timestamp should not be used as a reference to compare logs produced into different physical machines.

By properly interpreting each task, the developer can produce graphics similar to those frequently found in conventional simulations. For example, a task Y is designed to perform a payment through paypal in ebay after a successful auction. By accounting the times this task was executed, and using timestamps, it can be generated a graphic of the number of finished auctions through a determined period of time.

IV. THE CASE STUDY

Technological advances are increasing with time. The volume of scientific publications patents, research projects, technology news and related international standards of technology is in continuous increase. This makes available to researchers, R+D organizations, and industry in general, a huge amount of information to analyze for their projects and strategies. Technology Watch Systems are involved in processing of all information technology environment to extract knowledge, such as identifying trends and changes. This case study focuses on the management of quality of information sources within a Technology Watch System.

Technological watch is a tool used within *Competitive Intelligence*. Competitive Intelligence is the legal obtention, analysis, distribution of information about a competitive environment, including strong and weak points as well as the intentions of competitors [9]. Technological watch is an organized, selective and permanent process for information gathering scientific and technological information coming from in and out the organization; selecting it; analyzing it; distributing it; and communicating it; to convert it into knowledge supporting decision making activities with lower risk and being able to anticipate changes [10].

In an organization dedicated to R+D, clients of a Technological Watch are research groups. These researchers, generally, have already located relevant information sources to be watched. Therefore, researchers are a potential suppliers of information sources. If the system was feeded with bad quality information sources, the system would supply results with noise causing the analyses to be inadequate or wrong. This problem suggests the system should not accept all suggested information sources. In fact, giving more relevance to those

researchers investing effort in providing good information sources would potentially increase the quality of the produced results. Similarly, controlling more bad suggestors allows to keep the quality degree.

The development of Technological Watch system follows recommendations from the UNE 166006:2006 EX [10]. This normative provides a set of requirements, but no APIs or formal definitions. One of the recommendations consists in qualifying information sources with some attributes, which are highly related to the reputation and trust models well known in the agent literature. These attributes are supposed to be managed by humans, what means necessarily increasing the amount of work of operators.

Therefore, this case study to what extent reputation and trust models can be integrated in a Technological Watch System. The question is how such functionality can be integrated, i.e., are new components needed? is it enough with modifying the responsibilities of existing components?

This evaluation has required building a prototype dealing with three basic scenarios corresponding to the use case *Manage quality of information sources by means of reputation and trust models*:

- A low quality information source proposal made by a collaborator with low reputation in general
- A high quality information source proposal made by a collaborator with high reputation in general
- A low quality information source proposal made by a collaborator agent subject of bad reputation on behalf a supervisor agent which has witnessed past requests from the same agent.

The development made focuses in the first scenario.

V. DEFINING WITH INGENIAS

In the long term, the developed MAS aims to discover what kind of agents are required in order to have a population representative of a real scenario. Also, As it is now, it serves to experiment with the necessary protocols for integrating a trust model in the information sources management mechanisms.

As figure 1 shows, there are four groups of agents in the organization responsible of technological watch services.

- **Collaborators.** They are agents which propose new information sources. These agents can be a human operator representative or, directly, an agent that does not require a human operator.
- **Supervisors.** They are responsible of deciding how to evaluate proposals from the collaborators. The evaluation itself is performed by agents belonging to the Test Team.
- **Test Team.** They assign a quality value to an information source prior to its incorporation into the system. They do this by pretending the source is already incorporated and starting to use it with some predefined queries. Also they can request human expert evaluations.
- **Operations Team.** They watch accepted information sources and other technological watch services. There are agents within this group who are in charge of inspecting accepted information sources. They maintain

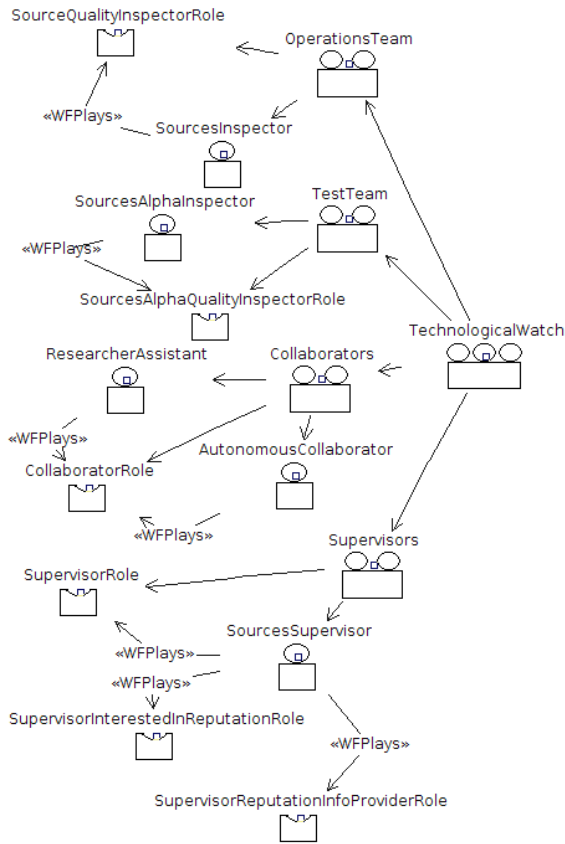


Fig. 1. Organization responsible of Technological Watch

updated information about the quality of the sources. This evaluation is used later to compute the trust degree of collaborators.

The MAS developed uses the REGRET trust model [11], making two simplifications. All supervisors have a credibility of 1 and only reputation information from witness will be taken into account.

Focusing on a *SupervisorRole*, this role has as goal keeping the system with high quality information sources. The capabilities of this role are expressed as tasks (fig. 2):

- **ProcessReceivedProposalTask.** The agent can process proposal from collaborators.
- **AddSourceIntoSystemTask.** The agent can add accepted information source into the system. This task also includes the request of quality inspection for the accepted information source.
- **RequestAlphaQualityInspectionTask.** The agent can request alpha quality inspection to inspectors. “Alpha quality inspection” means making a quality inspection without adding the information source into the system to be watched.
- **ProcessAlphaQualityInspectionResultTask.** The agent can process the results of alpha quality inspections.
- **ProcessQualityInspectionResultTask.** The agent can process the results of quality inspections.

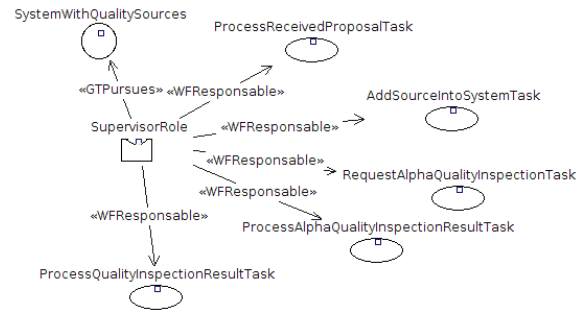


Fig. 2. Tasks assigned to a Supervisor

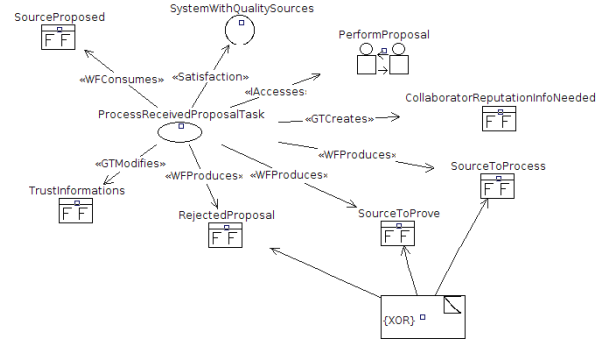


Fig. 3. Description of the task which process an information source proposed

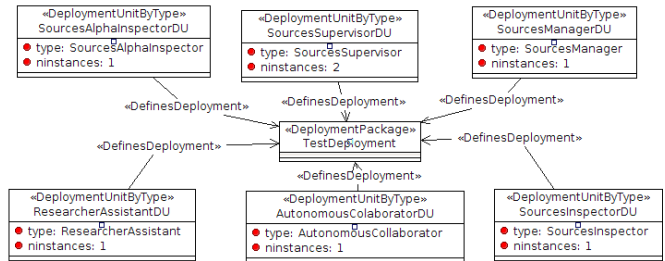


Fig. 4. Definition of the deployment for the test

Task *ProcessReceivedProposalTask* (fig. 3) is activated when the fact *SourceProposed* is found. The *SourceProposed* fact comes in the proposal message of the *PerformProposal* conversation. This task makes decisions about what filters apply to the proposal as follows:

- Reject the proposal because the collaborator has attempted too much to add information sources with low quality. The task produces the *RejectedProposal* fact to be sent as response in the *reject-proposal* speaking act of the *PerformProposal* conversation.
- Request an alpha quality inspection for the information source in order to obtain quality information in some criterias. It is because the supervisor doesn't trust in the collaborator about the quality (in some criterias) of information sources which he usually proposes. The objective behind is to apply the selected filters, that is, *If the information source has the quality value (in a specific criteria) less than the minimum quality value permitted (in*

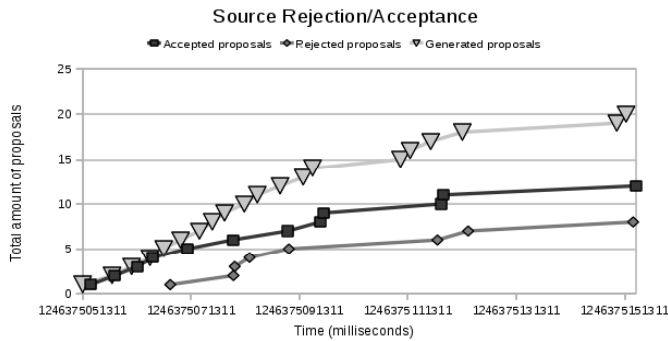


Fig. 6. Information source acceptance/rejection ratio

the specific criteria), then reject the proposal. Otherwise, accept the proposal. The task produces the *SourceToProve* fact to indicate the request.

- Request the adding of the information source into the system to be watched. The proposal is accepted because the supervisor trust in the collaborator. The task produces the *SourceToProcess* fact to indicate the request.

The *TrustInformations* fact has the information about trusting of all collaborators who have made proposal to the supervisor.

VI. SIMULATING

Figure 5 shows the GUI of the developed system where different actions can be triggered to run the basic scenarios. The log corresponds to these sequence of actions. First, the Collaborator makes a proposal of a information source with regular quality. The Supervisor #0 applies the filter, it's mean, request an alpha quality inspection to the Alpha Inspector. Also, The Supervisor #0 requests reputation information to the Supervisor #1, and the later response with the reputation information which has a high reliability. The Supervisor #0 accepts the proposal. Second, the Collaborator makes a new proposal with low quality. The Supervisor #0 doesn't trust (trust degree information with high reliability) in the Collaborator, then the agent decides to apply the filter to the proposal, and finds that the proposal has low quality and must be rejected.

The simulation of the system leads to a log of several megabytes of information. By filtering the content, and focusing in the production of *AcceptedProposal* entities, *RejectedProposal* entities, and *SourceProposal*, it is straightforward to obtain a list of events which tell when such entities are incorporated into each individual agent mental state. These entities are representative of a rejection, acceptance, and proposal of new information sources. Hence, accounting occurrences, one can determine the performance of the system. As figure 6 indicates, there are rejected sources and accepted sources. Hence, the rejection mechanisms are used. Nevertheless, it would be necessary to determine if the rejections should actually happen, something not accounted here.

VII. EVALUATION

After developing this prototype, a greater knowledge of the problem has been acquired. Using INGENIAS, generic information exchanges were depicted, detailing the information exchanged and having some wired code dealing with its transformation.

The development time was reduced to a minimum, one person for one weeks at full time (eight hours a day). Taking into account that the actual Technological Watch system has been developed for two years, this seems a reasonable price for having an accurate specification of the problem. Also, incorporating the produced system as an add on to the real system remains a possibility.

The simulation of the environment was straightforward to produce. Since the protocols were already established, it was known what information the system was expecting from the users. So, defining GUI agents interfacing with real human operators or agents pretending to act directly with the system was easy. From here, deciding the kind of agents to deploy and their specific features, was a matter of depicting an ingenias deployment entity.

The degree of reusability is not known yet. The model was concluded recently and it is currently under the evaluation of other partners in the project. Should the specification be accepted, the ideas would be incorporated into production, modifying the current Technological Watch system being used.

VIII. RELATED WORK

The current state of art of agent oriented software engineering methodologies shows only a little number of methodologies permitting to produce directly code and perform the kind of simulations made with INGENIAS. Methodologies like MaSE [12] or Prometheus [13] are capable of code generation. Nevertheless, they intend to produce fully functional systems everytime and do not conceive the use of simulations as part of the development. In the case of Prometheus, code generation is a recent incorporation so its effective use to produce MAS automatically is still under study (there is a plugin but the documentation for its use has not been update as of today). In the case of MaSE, code generation has been integrated since the methodology was born. Systems are specified almost completely from the tool. Nevertheless, the customization of the produced code is not as effective as in INGENIAS. A developer in INGENIAS will find several tools to synchronize the produced code with the specification. In MaSE, this possibility does not exist.

ADELFE [3] bases on several simulation platforms, one of the most recent is SeSAM. The behavior of agents within SeSAM is made by means of activity diagrams, permitting the developer to express a variety of possible agents. Nevertheless, the production of a SeSAM specification is achieved manually using ADELFE concepts. This complicates the synchronization of both the problem specification and the simulation, something that does not occur with INGENIAS and the IAF. Besides, the effort for defining an agent is lower in INGE-

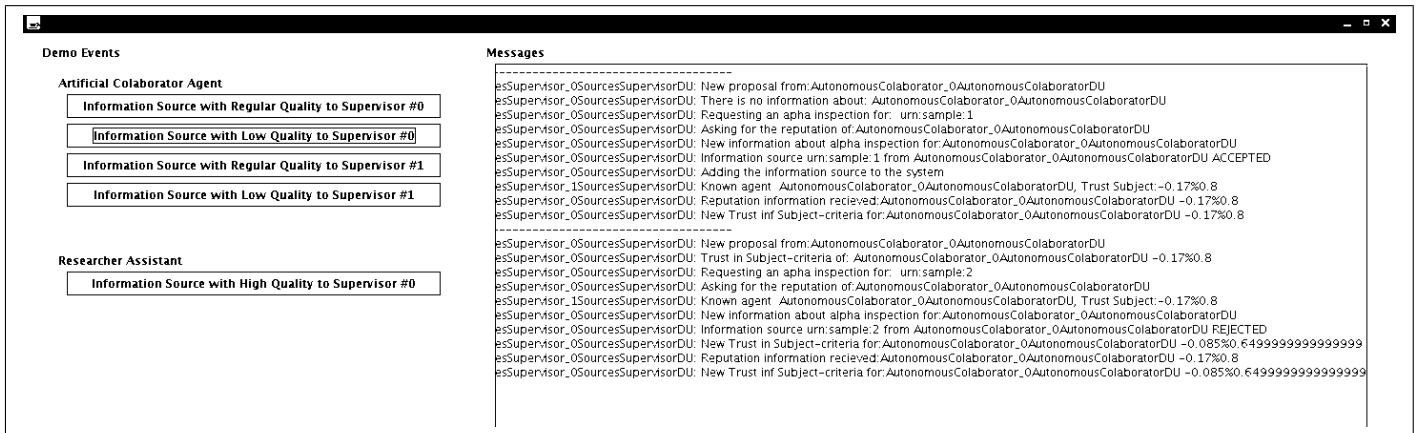


Fig. 5. Agent learns not to trust in another agent by direct experience and reputation

NIAS. Doing the same in SeSAM implies manually modifying the SeSAM agent template to incorporate the activities.

PASSIM [4] uses state-chart based simulation to validate and produce prototypes. The formalism used to describe the system to simulate is Distilled StateCharts. The translation between design concepts and simulation concepts is semi-automatic. There is a first stage which produces the skeleton and a second stage that requires human intervention to refine the code. Simulation in PASSIM concerns the whole system. In the work introduced in this paper, the simulation generation is automatic and its content can concern the whole application or only its environment. Also, INGENIAS provides means to integrate with external applications, where PASSIM does not. INGENIAS simulation agents base on the BDI paradigm and are coded that way. In PASSIM, the coding corresponds to the statechart formalism. Like in SeSAM.

IX. CONCLUSION

Prototyping and simulations are two ways of clarifying system requirements and experimenting with different approaches in a domain problem. Prototypes are expendable and simulations, depending on the support tool, are expendable as well. A simulation performed with SeSAM, for instance, cannot be used as a final product to be delivered to end users. Hence, an approach permitting prototyping, creating simulations, and reduce costs in producing both, would be welcome.

INGENIAS can provide such services. In INGENIAS, with the aid of the INGENIAS Development Kit and the INGENIAS Agent Framework, it is possible to experiment different configurations of a MAS investing little effort. Also, it is possible to create artificial environments where there are simulated human operators or external agents interacting with the developed system. The result of the experimentation is a MAS specification capturing the requirements of the client, whose interpretation can be inspected visually by the client; and a MAS obtained automatically from the specification, which can be used as prototype or as final system, depending on the needs of the development.

ACKNOWLEDGMENT

We acknowledge support from the project *Agent-based Modelling and Simulation of Complex Social Systems (SiCoSSys)*, supported by Spanish Council for Science and Innovation, with grant TIN2008-06464-C03-01. Also, we acknowledge the funding from the *Programa de Creación y Consolidación de Grupos de Investigación UCM-Banco Santander* for the group number 921354 (GRASIA group).

REFERENCES

- [1] M. Schrage, "Cultures of prototyping," pp. 191–213, 1996.
- [2] M. Luck, P. McBurney, and C. Preist, "A manifesto for agent technology: Towards next generation computing," *Autonomous Agents and Multi-Agent Systems*, vol. 9, no. 3, pp. 203–252, 2004.
- [3] C. Bernon, M. P. Gleizes, and G. Picard, "Enhancing self-organising emergent systems design with simulation," in *ESAW*, ser. Lecture Notes in Computer Science, G. M. P. O'Hare, A. Ricci, M. J. O'Grady, and O. Dikenelli, Eds., vol. 4457. Springer, 2006, pp. 284–299.
- [4] M. Cossentino, G. Fortino, A. Garro, S. Mascillaro, and W. Russo, "Passim: a simulation-based process for the development of multi-agent systems," *IJAOSE*, vol. 2, no. 2, pp. 132–170, 2008.
- [5] J. Pavón and J. J. Gómez-Sanz, "Agent oriented software engineering with ingenias," in *CEEMAS*, ser. Lecture Notes in Computer Science, V. Marik, J. P. Müller, and M. Pechoucek, Eds., vol. 2691. Springer, 2003, pp. 394–403.
- [6] J. Pavón, J. J. Gómez-Sanz, and R. Fuentes, "Model driven development of multi-agent systems," in *ECMDA-FA*, ser. Lecture Notes in Computer Science, A. Rensink and J. Warmer, Eds., vol. 4066. Springer, 2006, pp. 284–298.
- [7] J. J. Gómez-Sanz, R. Fuentes, J. Pavón, and I. García-Magarino, "Ingenias development kit: a visual multi-agent system development environment," in *AAMAS (Demos)*. IFAAMAS, 2008, pp. 1675–1676.
- [8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, January 1995.
- [9] K. Contrill, "Turmin Competitive Intelligence into Business Knowledge," *Journal of Business Strategy*, vol. 19, Julio/Agosto 1998.
- [10] AENOR, "UNE 166006:2006 EX: Gestin de la I+D+i: Sistema de Vigilancia Tecnolgia," UNE, Final, 2006.
- [11] J. Sabater, "Trust and Reputation for agent societies," PhD Thesis, Universitat Autnoma de Barcelona, 2003.
- [12] J. C. Garcia-Ojeda, S. A. DeLoach, and Robby, "agentool iii: From process definition to code generation," in *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems*, 2009, pp. 1393–1394.
- [13] L. Padgham, J. Thangarajah, and M. Winikoff, "Auml protocols and code generation in the prometheus design tool," in *AAMAS*, E. H. Durfee, M. Yokoo, M. N. Huhns, and O. Shehory, Eds. IFAAMAS, 2007, p. 270.

Multiagent Simulation Model Design Strategies

Franziska Klügl

School of Science and Technology

Örebro University, Örebro, Sweden

Email: franziska.klugl@oru.se

Abstract—Model design is particularly challenging for multi-agent simulation models as the simulation paradigm does hardly impose constraints on it. This contribution systematically analyzes procedures for developing a multi-agent simulation model: iterative methods derived from principles, such as KISS or KIDS and methods focussing on the different design elements (agents, interaction, environment).

I. INTRODUCTION

Methodological questions are more and more in the focus of research on agent-based simulation. This contribution deals with the phase in modeling that requires most experience and creativity from the modeler: model concept and design.

The development of a multi-agent simulation model seems to be an inherently intuitive way of modeling: Entities observable as active in the real system are captured as active entities – as agents – in the model. There seems to be no need for complex aggregation or abstraction necessary for formulating the model, nor a languages that requires deep mathematical skills must be used. Features of the model such as variable structure, heterogeneities, mixed local and global effects, ..., can just be formulated or generated from more or less simple agent behavior and interaction – just as observable in the original system or stated in the original ideas. Due to this intuitiveness and its general potential, it is often forgotten, that the development of a multi-agent simulation model leads to particular problems beyond general simulation engineering:

- Determining the appropriate level of detail is everything else but trivial. Actually, it is the basic design problem to determine what behavior shall be included or which factors ignored.
- Thinking in terms of agents can also be a problem when the modeler is used to other paradigms, such as process-oriented or macro modeling. Then an interaction-oriented model for generating the aggregate behavior instead of describing it, may be difficult to design.
- The general intuitiveness of the modeling leads to a tendency of ad-hoc development. This is supported by visual programming tools such as SeSAM [1]. Modelers immediately start implementing instead of thinking about an appropriate design beforehand.
- Necessary effort for understanding and analyzing the model and its overall behavior is underestimated. As in a multi-agent simulation the overall behavior is generated from the interactions and local behavior of agents, special effort has to be invested for excluding artifacts originating

from minor bugs at the micro-level – in the agents design as well as implementation.

- A last issue is the difficult control of the included assumptions – due to the size of the model and the effort of developing, a systematic control of the assumptions taken while modeling has to be done. After modeling all assumptions can hardly be recapitulated and tested when they are not explicitly collected while modeling.

These are issues mostly in design of a multi-agent model. Clearly there are others, when considering all phases of the simulation study. Examples for additional problems are the need for validation when relevant data is missing, the difficulties of implementing concurrent processes or technical problems due to the size of the model and simulation. Nevertheless the design phase of a modeling and simulation study is the one that requires the most experience. This is the phase that is often coined as “art” [2].

In this contribution, we are focussing on the issues in model design and give a systematic view on different procedures. In the following we will first set the context of our endeavor by tackling process models for (multi-agent) simulation, as well as input from agent-oriented software design. Then we will give procedures for designing a model – one-shot in section III and iterative in section IV. The contribution ends with a discussion of the procedures and a short conclusion.

II. METHODOLOGIES AND PROCESSES FOR DEVELOPING MULTI-AGENT SIMULATION MODELS

A. Simulation Study Life Cycle

Phases of a systematic development of agent-based simulation models have been suggested by several researchers [3], [4], [5]. While focussing on integration of data or different roles of participants in the study, their suggestions for procedures are naturally quite similar to guidelines for general simulation study development developed by [6], [7] or [2]. The basic phases are initial works such as fixing the objective, making a profound analysis of the system or gathering necessary data and information. This is followed by making a model concept and elaborating it - this is basically model design. Model implementation and calibration are additional phases that are accompanied by analysis and validation phases. When the model is ready, the simulation study is completed by deployment runs and documentation and interpretation of the results. Clearly, bugs and deficiencies discovered in one phase may require re-working in previous phases. Figure 1 depicts

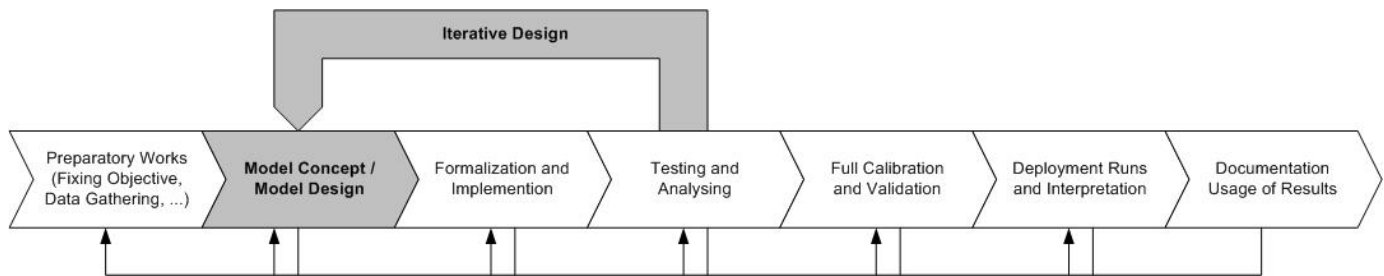


Fig. 1. Phases of an agent-based simulation study. Model design may also be iterative.

these basic phases for agent-based simulation development. The model design phase is highlighted.

It is quite unrealistic to assume that model design can be finished within one cycle. An iterative procedure for developing the model is appropriate not only for finding the right level of detail – starting with a initial model that is developed further until it is run-able and its dynamics analyzable. Based on the insight from this analysis, the model is further improved. Such an overall iterative proceeding is indicated in Figure 1.

This process is very abstract and does not use abstractions and techniques that are particular for agent-based simulation. This would be necessary for a more specific simulation life cycle. In contrast to agent-based software, there is no fully elaborated meta model describing necessary elements of an agent-based simulation in general (or even for a particular domain) in sufficiently concrete detail. Thus, a methodology comparable to the ones suggested for agent-based software engineering cannot be given yet. First attempts for a formal meta model for multi-agent simulation models have been made, such as in [8], but they are not on a level that makes them usable as a basis for the development of more specific life cycle models.

Before we will tackle model design and possible iterative improvement strategies, we will first have a short look on agent-oriented software engineering where the design of an agent system is central part of every methodology.

B. Agent-based Software Engineering

For the development of agent-based software, many methodologies have been suggested. Clearly, they are based on general software development phases such as requirements engineering/analysis, specification/design, etc. These methodologies rely on appropriate abstractions for representing and analyzing the agent-based software system. Development then happens by elaboration of these representations. Many methodologies have been proposed for guiding the development of agent-based software based on roles or organizational structures or focussing on specific agent concepts and architectures, such as BDI. Good collections can be found for example in [9] or [10]. Comparisons based on benchmarks [11] or features [12] aim at supporting the selection of the appropriate methodology for a particular problem. We would just like to mention [13] showing how modeling and simulation can be used for the design of self-organizing agent software. The suggested

procedure has some similarities with the work described in this paper; yet here the focus lies in simulation applications with their specific characteristics. Our goal is to deal with a guideline how to modify the different elements of a model to finally build a valid simulation model.

Although there are profound differences between agent-based software and agent-based simulation – such as the correspondence to an original system ensured by validation or the inclusion of a simulated environment in addition to the simulation environment – the abstractions used for designing the software system may also be useful for designing a simulation model. This is especially the case when real-world organizational structures are used or folk psychology-based agent architectures are appropriate in the simulation model. In the following we want to tackle strategies for simulation model design from a simpler point of view:

III. AGENT-BASED SIMULATION MODEL DESIGN

In the following we will introduce and discuss three design strategies. They are derived systematically from an idea about elements of multi-agent simulation models: It consists of three basic components: agents model, environment model and model of the interactions. Each of them is used as a starting point or driving force for a design strategy; thus their usage can be seen as exclusive. However, it might be advisable to use different approaches one after the other in an iterative setting.

A. Agent-driven Model Design

“Agent-driven model design” is the first strategy. It corresponds to bottom-up design. The focus lies completely on the agents, their decision making and their behavior. Environmental and interaction models are added when needed in the agent design.

1) *Basic Strategy*: The following procedure can be defined:

- 1) *Agent observation and coarse behavior description*: The modeler observes the real-world agents and derives a coarse behavior description from its observations. Observations may be replaced by literature work or operationalization of hypothesis about agent behavior/decision making.
- 2) *Categorize agents and determine the location of heterogeneity*: The coarse behavior descriptions are analyzed for determining how many classes or types of agents are necessary for the model and to what level the agents

should be different. The location of heterogeneity may be on the level of parameter settings, different activities or even completely different classes.

- 3) *Decide about agent architecture:* Based on the coarse behavior description that treated the agents superficially, the modeler has to decide about the architecture of the agent. He may select a behavior-describing approach, for example perception-action rules with or without internal state representation or an architecture that is explicitly grounded on goals and on the configuration and selection of plans or even using plan generation from first principles or an elaborated cognitive model. This selection is depending on the complexity and flexibility of necessary agent behavior.
- 4) *Formalize agent behavior/goals:* The next step consists of filling in the actual behavior into the agent architecture. This is done by analyzing, elaborating and refining the coarse behavior description developed in the first steps.
- 5) *Add interactions and environmental aspects when needed:* Particular elements of the environmental model or structures of interaction are added when the agent behavior needs to include particular perceptions, messages or contains manipulations of environmental entities. As these aspects are added in some ad hoc manner, some re-factoring may be necessary, such as merging different environmental entity classes. Also some considerations about necessary heterogeneity are essential.
- 6) *Test whether necessary macro-phenomena are sufficiently reproduced:* At any time, model validity has to be tested when it is testable. We assume that the focus on the agents will lead to agent behavior near validity. But, a major effort will be testing whether the interplay of agents and their environment results in a valid macro-level behavior.

This procedure is a pure agent-driven bottom-up method for developing a multi-agent simulation. Aspects that relate agents to others or the environment are only important in so far they are influencing the agent behavior. Before we continue to discuss this procedure, we give a short example how the application of this method may look like.

2) *Example:* We used an existing simulation model[14] of bee recruitment: The basic objective was to find support for the hypotheses that the environmental structure influences the success of a recruitment mechanism in social insects.

Using an agent-driven approach, this model is build from the point of view of a honey bee. Individual bees are to be observed and literature has to be scanned for description of behavior and parameter values. In the second step, different categories of bees are identified: scouts, foragers or bees waiting/working in the hive. In this particular case these categories correspond more to activities than to different agent types as agents may instantaneously switch between activities. Therefore one may decide for a homogenous agent population with differences in the currently executed activity.

The following tasks of an agent bee are to be considered:

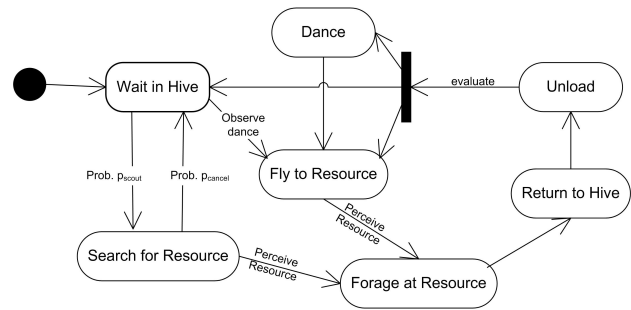


Fig. 2. Specification of example bee-agent behavior using activity graphs (adapted from [14]).

scouting, foraging, returning to the nest, unloading, wagggle dance (recruiting) and waiting. A simple behavior-descriptive architecture is appropriate. Due to the identified activities, an approach based on activity graphs representing the relations between activities is useful.

The next step is the formulation, i.e. specification and implementation of agent behavior. In figure 2 we depict the behavior of an agent.

Figure 2 does not indicate, when an interaction with the environment or other agents has to take place. When elaborating this graph, the modeler concurrently determines that there is a 2dimensional map for scouting and discovering resources. There must be resources that provide a nectar of certain quality. When returning to the hive, there must be some place to unload and some place to dance for recruitment of others. The storage area and the dance ground are only dealt with on a very abstract level. They may only be attributes of an object of the type "hive". For switching activities between waiting and foraging, the actual recruitment has to be modeled. Information is displayed by the dancer and perceived by conceptive other ants that based on the received information decide to scout or not.

This specification has to be implemented and tested as described above. As aggregate measures, the nectar input may be available for macro-level validation as well as counts how many bees are dancing, how many fly out, etc.

3) *Discussion:* The result of this process is a model that is fairly process-oriented on the agent level. For the example, the agent-driven design seems to work quite well. This might be due to the fact that interaction happens only indirectly via the environment or by displaying or broadcasting information. In the example, there is no direct message-based interaction. In principle the procedure should also work with direct peer-to-peer interactions. However, the strategy does not contain any perspective on the system-level containing for example protocol specifications. Such a bottom-up technique where the modeler takes over the perspective of an agent can be appropriately tested using participatory simulations where one agent is controlled by a human, the others are simulated.

A critical issue occurs when the overall validity is not reached. Then, this pure bottom-up technique will lead to trial and error procedures as during the development of the model

no connection between macro- and micro level is established. Additionally this procedure does not help in finding the appropriate level of detail. Therefore it needs to be combined with an iterative procedure.

B. Interaction-driven Model Design

There are simulation domains where a focus on interactions is more appropriate than a purely agent-driven design. Examples may be models that focus on the performance of organizational design. In the following we want to introduce interaction-driven design.

1) *Basic Process:* One can formulate the following basic process for interaction-driven model design:

- 1) *Identify actors/entities and interactions among them:* Instead of observing individual real-world agents, the modeler is taking the birds' perspective and analyzes who is interacting and how.
- 2) *Coarse description of protocols and their conditions, constraints, etc.* The identified actors and interactions are refined to protocols going from general notions of interaction to atomic exchanges of information and manipulations of the environment.
- 3) *Derive agent behavior* for producing the interaction elements (messages, signals, actions...) and add environmental entities, such as shelter objects, to the model when needed for interaction. In this step something like a finite state machine based language can be used to specify agent interactions as state transitions.
- 4) *Implement agent behavior and test* whether the intended interactions and their outcome on the macro level are actually produced by the overall system. It must be also tested whether the agent level behavior is plausible or valid – depending on the available data.

In the interaction-driven approach, agents are basically seen as black boxes for producing the appropriate messages, information, etc. The general procedure may be further developed into some form organization-driven model design inspired by similar methods and methodologies from agent-oriented software engineering. Then, analysis of organizational structures forms the starting point for all system analysis as it forms the structural backbone of interactions.

2) *Example:* We use the same example as above for illustrating the approach and its differences to the agent-driven procedure.

As given above, we start by identifying the actors and their interactions. Again we have to tackle the problem of the location of necessary heterogeneity. Are the actors to be found on the level of scouts or foragers or as bees on a more homogeneous level? In our particular case we took bees as agents. Table I shows the basic interaction between the different types of entities.

The next step is the elaboration of the protocols. We suggest to use UML-based interaction diagrams for the initial description. In figure 3 we show the description of the recruitment protocol together with an (still) informal text about its context of appearance.

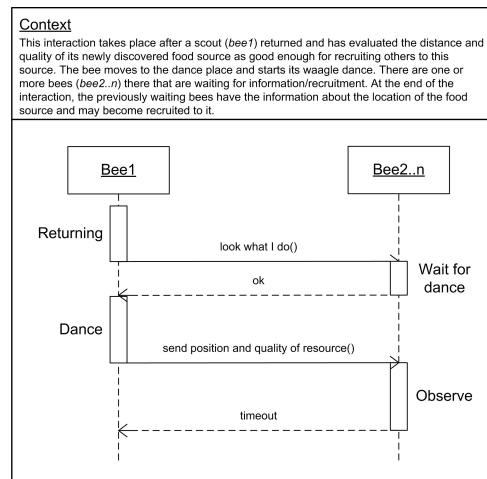


Fig. 3. First specification of recruitment protocol together with context description.

Formulating this interaction is not trivial as it is more like a broadcast (or stigmergic interaction). The message is send – respectively the information is displayed in the dance – to all agents that want to listen or observe it.

The next step would be to derive agent-behavior from the interaction diagrams. We suggested to transfer the interaction diagram into some finite state machine like representation. If we do that for every interaction the agents are participating in, a collection of finite state machines is generated. These single simple graphs have to be combined into a complete behavior description. This is done by first identifying similar states as interfaces which's unification merges the single state machines into some larger one. This larger one is probably too large as every interaction aspect is modeled explicitly. It might be possible to simplify some parts. Therefore, a refactoring might be necessary for bringing the overall state graph into some well-structured and minimal form.

Figure 4 shows the straight forward combination of a number of single-interaction state charts. For connecting we identified two times similar nodes – end nodes of one interactions, starting nodes of the other. In the graph such nodes are depicted as black nodes with second ring.

During the development of this behavior graph, we had to add state machines for interaction partners that are not given in figure 4. These interaction partners are the bee hive sending information that is used for the evaluation of the value of the load when the bee-agent is returning to the hive. Additionally, we omitted the interaction with the overall environment where the agent is requesting and receiving perception information and with resources with that it interacts while harvesting the nectar of this resource.

As a next step, it has to be determined what happens within the different states. This is straight forward, often consisting of waiting for incoming messages.

3) *Discussion:* Despite of the birds' eye perspective, this procedure did not result in an abstract and minimal behavior description. Nevertheless, the interaction and dependency of

Interactions	Bees	Resources	Nectar Storage
Bees	Recruitment	Harvest	Unloading
Resources	Localization	-	-
Nectar Storage	Status Information	-	-

TABLE I
INTERACTION TABLE FOR BEES AS AGENTS.

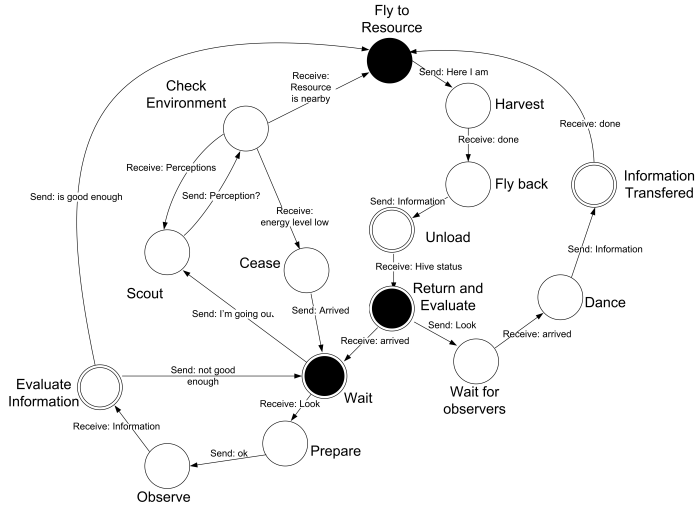


Fig. 4. Putting together single interaction state machines into a complete agent-level graph that describes all interactions of an agent. The black nodes were the initial start and end nodes of the two graphs. They are basically the interfaces between the two state machines .

behavior on information and material provided by others is treated explicitly, much more than in the agent-driven approach. This is actually an advantage of this procedure.

However, one can foresee problems when actual pro-active behavior has to be formulated. That means behavioral dynamics that are not triggered by external messages. Another drawback is, that also resources and other entities that are basically are no agents, have to be treated as agents as every interaction is formulated based on protocols. Although these are then just “passive” agents, they have to be practically tackled as agents for specifying the interactions.

Due to its focus on direct interaction, other forms of interaction may be hard to model, such as stigmergic interaction in form of broadcasted messages that are persistent in the environment decoupling sender and receiver. However, for simulated multi-agent systems with interaction that relies on message-based communication this design strategy seems to be appropriate.

C. Environment-driven Model Design

The third strategy we want to analyze is driven by a focus on the environment.

1) *Basic Process:* In analogy to the previously discussed design strategies, the starting point of the environment-driven design is an analysis of the environmental structure. Based on this, the agent interface and its behavior definition are determined. The steps are in particular:

- 1) *Identify relevant aspects* (global status, global dynamics/ local entities) of the part of the model that represents the environment.
- 2) *Determine the primitive actions of the agent and the reaction of the environmental entities to these.*
- 3) *Determine what information from the environment must be given to an agent so that it can appropriately select and perform its actions.*
- 4) *Decide on an agent architecture* that is apt to connect perceptions and actions of the agent appropriately for actually producing the agents behavior. Concurrently, the elements of the internal agent status are settled.
- 5) *Use a learning mechanism for determining the actual agent behavior.* A reward function for providing feedback to the agents actions has to be found. The reward schema also tackles questions such as when and how often to provide feedback to the agents, whether all agents learn based on a shared reward or individual reward is given to the agents.
- 6) *Implement the environmental model* including reward function if needed.
- 7) *Specify and implement the agents behavior program* or agent interfaces in combination with the chosen learning mechanism.
- 8) *Test and analyze the overall simulation* results and individual trajectories carefully for preventing artifacts that come from an improper environmental model or weak interfaces (perceivable situations and effects of primitive actions).

2) *Example:* For an illustration of this model design strategy we again use the recruitment scenario although the environmental complexity is not high enough to actually require such a procedure.

The start is made by formulating the environmental model. In this simple case the environment consists of a global world entity managing a 2-dimensional map, a central hive entity that is basically a container for the storage and a number of resource entities randomly distributed over the map, each with an individual nectar supply.

The initial environmental configuration in this case is the following: the hive is positioned in the center of the map. Each resource is located at a random position. Every resource object has an attribute called “nectar supply” that is initially set according to a normal distribution.

The next step is to design the perception capabilities and possible primitive actions – the interfaces of the bee agents. Without particular regard on what is actually needed in the behavior definitions, we can list the following perceptions:

- perceive nearby resource, its position respectively (if nearby)
- perceive existence of resource (from a certain distance)
- perceive capacity of resource (if nearby)
- perceive hive storage (if nearby)
- perceive position display (if at hive)
- perceive current nectar load

... and actions:

- Perform random search
- Fly towards perceived resource
- Fly towards hive
- Load nectar
- Unload nectar
- Display resource information
- Memorize perceived position

One can notice that with defining this interface, the modeler also determines the abstraction level of the behavior definition – the environmental model alone did not fully determine the level of abstraction.

The next step is to connect perceptions and action to produce actual agent behavior. In our case this could be done using a rule-based approach with rules defined by the modeler. The simple interface that already indicates that a rule-based approach – including some stochasticity in agents decisions for some very abstract treatment of internal motivation – might be sufficient.

We may suggest the following rules determining the agent behavior:

- 1) *if* hive-storage < A *then* perform random search (with probability p_A)
- 2) *if not* at hive *and not* perception of resource *then* perform random search
- 3) *if* perception of resource *then* fly towards perceived resource
- 4) *if* at resource *then* memorize resource information
- 5) *if* at resource *then* load nectar with rate *load*
- 6) *if* nectar load > B *then* fly towards hive
- 7) *if* at hive *and* nectar load > B *then* unload nectar with rate *unload*
- 8) *if* at hive *and* resource information memorized *then* display resource information
- 9) *if not* at hive *and not* perception of resource *then* fly to hive (with probability p_{cancel})

This set of rules is quite small, but sufficient. There are some probabilities and thresholds to be set ideally based on available data.

While not necessary in the example, using a learning mechanism might be an appropriate solution for generating the behavior program based on the perceivable situations and primitive actions of the agents. A learning approach, e.g. based on classifier systems seems to be feasible at least in this application example.

The major question in this application example is when to give feedback as an information to the agent about its performance: Giving feedback after each step does not make sense: the random search without information is intentional. The agents shall not learn where the resources are positioned. Ideally, positive feedback shall be only given when the agent has accomplished to recruit other bees to a good resource or even more delayed, the reward to all agents could be proportional to the current influx to the overall hive storage.

3) *Discussion:* Again one can find potential problems considering the agents internal motivations for generating true pro-active behavior without external triggers. Such elements of

complex agent-based simulation models are not well integrated into this design procedure.

Involving learning mechanisms forms a basis for research questions with evolutionary background. However, integrating agent learning into the model design makes the model susceptible for artifacts coming from incomplete, imprecise or not fully elaborated reward or fitness functions. Agents that adapt to an environmental model with flaws can never reliably reproduce an original system independent how good the learning mechanism or the rest of the model is. Nevertheless, it is not trivial to find appropriate feedback functions characterizing the goal of the agents' development.

Another risk of this environment-driven approach consists in the selection of the learning mechanism. It could easily happen that no appropriate learning mechanism exists that can be used without further research. It is not so difficult to reach the boundaries of what is possible using current state-of-art learning: involving co-adaptation, true multi-agent learning with more than a couple of agents, non-Markov settings, etc. Then the learning problem may be too hard for finding a mechanism that converges within a feasible time.

IV. STRATEGIES FOR ITERATIVE MODEL DEVELOPMENT

The strategies introduced above are for designing an agent-based simulation model in one step. Especially for identifying the appropriate level of detail iterative procedures may be combined with these strategies.

A. KISS: “Keep It Simple, Stupid”

The well known KISS principle for simulation modeling means that the modeler should avoid unnecessary complex models, but keep the model as simple as possible for generating the appropriate behavior. Simple models contain less sophisticated assumptions, can be easier explained and understood. Simplicity also refers to the modeling and simulation paradigm used formulating and simulating the model.

The starting point of this procedure is the identification and description of phenomena of the original system that should be contained, respectively reproduced by the final model. Grimm and Railsback [15] call these basic modules of system data characteristics “pattern”. Examples for such pattern are a certain statistical distribution of tree sizes in a forest, another pattern is then the spatial distribution of large trees.

- 1) Identify and describe the set of observable properties (statements) about the real system S .
- 2) Define a model M_0 that is apparently too simple for reproducing the system with all its properties
- 3) By calibration, determine the set S_M of properties, that are reproduced by M_0 .
- 4) $M \leftarrow M_0$
- 5) Repeat Until $S_M = S$
 - a) $M \leftarrow$ modify model M for producing more elements in S than in the last iteration.
 - b) Calibrate M and determine S_M as the set of properties reproduced by M .

When this algorithm stops, the result should be the simplest model that captures all phenomena identified at the beginning of the process.

However, it remains open how to modify and enlarge the model for producing the next model from the previous one. Sometimes also side-steps might be necessary removing one model component and adding an alternative one. Finally, it is not trivial to see how to modify a model best for producing any additional phenomena. In worst case, this might result in a try-and-error procedure. Nevertheless the documentation of every single model shall be elaborated that especially contains a list of taken assumptions.

B. KIDS: “Keep it Descriptive, Stupid”

In 2004, B. Edmonds and S. Moss published a plea against in their eyes over-simplified models in social science (see [16]); Their major argument was: “The point is that it is simply not appropriate to make simplifications *before* one knows what is relevant and what not.” (italics in the original). They therefore suggest to initially construct a model with agent behavior that is understandable and directly deducible from the observed behavior, but not necessary simple.

The iterative algorithm based on this KIDS-principle based strategy can be formulated as in the following.

- 1) Repeat until a valid model M_s is constructed
 - a) Define a model M that contains all apparently relevant aspects of agent behavior.
 - b) Identify all assumptions and make explicit all parameter in M_i .
 - c) Execute a sensitivity analysis for all parameter of M and eliminate all blocks of behavior that are controlled by a parameter without effect on the overall outcome. M_s is the model M after sensitivity analysis
 - d) Test M_s for credibility and validity

At first sight, this procedure basically resembles the usual try-and-error strategy. It does not give any hint what to do if the model is not sufficiently valid in terms of aspects that are not reproduced; As it is less systematic than the KISS strategy, the KIDS strategy is more apt for experienced modelers that know with what model to begin and how to operate if the outputs are not as intended.

C. TAPAS: “Take a previous model, add something”

A third strategy for model design is pragmatic and focusses on reuse of models. In the agent-based simulation area, it has been coined by [17] without further discussing the term. It is related to the KIDS-based strategy but takes an existing model as starting point.

Reuse of models becomes the more inevitable the more expensive investments have been already made to produce the model. As construction and testing of an agent-based simulation is more expensive than for traditional analytical or simpler microscopic models, reusing a fully validated model should be especially interesting for agent-based simulations.

Put into a more pseudo-code way of presentation, the TAPAS strategy might look like the following procedure.

- 1) Select an *appropriate* existing model M
- 2) if M is not implemented, implement it and validate it using model alignment with respect to published data about M .
- 3) Add new, additional aspects to produce M_{add}
- 4) Test and Validate M_{add} , if sufficient, ready, else go back to 3 or - if necessary to 1.

Step 3 and 4 are similar to the KIDS methodology. The critical step is the selection (and existence) of the reusable model. A sufficient documentation of the original model is essential and can be seen as a major prerequisite for reuse [18].

There are also some perils: it is not known a priori whether the model with modification is minimal, valid, or possesses the intended properties. The advantages of having a starting point have to be traded off the effort that it means to adapt the given model to the new ideas. This is very risky when the model for reuse is not fully understood by people that want to reuse it.

Given appropriate tool support, also partial models, that means single agents, groups of agents or the complete environmental structure, may be used as a starting point for new model development.

D. Candidate-based Modeling

In his book about biological modeling in general Haefner [19] suggests a procedure for modeling and simulation in research. Basically, it consists of the construction of a set of alternative models. They might differ in parameter settings, but may also use different architectures, etc. Each of the model candidates is calibrated and evaluated by validation. The “best” model is selected and used as a basis for future research. This procedure is hardly about iterative changes, but involves the treatment of several models which has to be done when for example conflicting hypotheses have to be evaluated for possible rejection. The candidate-based strategy is generally applicable, but does not state how one may come to one candidate or from one candidate to another.

E. Discussion

In this section we surveyed iterative model design strategies. We did not tackle particular design strategies for one model, but general procedures how to proceed from a first prototype to the model ready for deployment. Which of these strategy is appropriate for a particular simulation study is depending on the personal style of the modeler, on his experience, on the application domain, on the available data, etc. The table II sums up and contrasts properties of these strategies. First, the appropriateness of the strategies for different types of multi-agent simulation models is estimated. A second block refers to directed-ness of the model and minimality of the outcome of the modeling process. The third block of rows compares aspects of necessary data availability and other properties of the modeling process - for example how well this procedure

Strategy	KISS	KIDS	TAPAS	Cand.-based
Apt for linear models	high	high	high	high
Apt for emergent phenomena for shared-environment actors	low mid	high high	high mid mid	high high high
Objective-orientedness	high	mid	mid	mid
Resulting minimality	high	mid	low	mid
Share of try & error	low	mid	high	high
Empirical data requirement	mid	mid	low	high
Integration of knowledge	mid	high	high	mid
Communication support	mid	high	mid	mid
Modeling overhead	mid	mid	low	high
Expertise in macro models	high	low	low	low
Expertise in micro models	high	high	low	high
Required tool knowledge	high	mid	mid	high

TABLE II
COMPARISON OF ITERATIVE MODELING PROCEDURES

supports the communication of the current model status. The last block refers to whether the modeler needs expertise, whether he has also to tackle macroscopic relations within the model – usually expressed in complex formulas, or whether this iterative procedure is also apt for beginners in agent-based modeling. We assumed that tools are available that support each of the strategies.

Using table II a modeler may select a specific iterative strategy for finding the best model in accordance with features of the simulation problem. Selection may also rely on common sense heuristics: if a good previous model is accessible, then it is a good idea to reuse that model. If the modeler knows the system that has to be simulated very well, then the KIDS approach should be used, whereas large holes in the system knowledge advise more candidate-based or KISS modeling. All these iterative procedures may be combined with the particular design strategies identified above.

V. CONCLUSION

Model design is the phase in the development of a simulation model, that requires most experience and skills[2]. This is true for all forms of simulation modeling, in particular for the development of multi-agent simulation models. In this contribution we have identified and discussed different strategies for model design – one shot or iterative. The latter can be combined with the former for really developing a well designed model. Even, if the resulting model is not significantly “better” than a model developed without these strategies, their application has the great advantage that they support a systematic development and at least partially guide the development. Nevertheless, the different strategies have to be further tested and elaborated in various simulation studies with different degrees of necessary model complexity.

The vision behind our research is developing a methodology for successful development of multi-agent simulation studies. The idea is to develop a methodology similar to *System Dynamics* [20] that allows the development of difference equation models starting from causal loop graphs to systems of formulas in a guided and systematic way. In agent-based simulation, systematic model design has to be accompanied with

equally thoughtful implementation, calibration, documentation and especially validation. We already tackled these phases in an insolated way that leaves a lot of research open for combining the different solutions. This – and the application and test of the identified design strategies – form the next steps in the future.

REFERENCES

- [1] F. Klügl, “Sesam: Visual programming and participatory simulation for agent-based models,” in *Multi-Agent Systems: Simulation and Applications*, A. M. Uhrmacher and D. Weyns, Eds. Taylor & Francis, 2009.
- [2] R. E. Shannon, “Introduction to the art and science of simulation,” in *Winter Simulation Conference 1998*, 1998, pp. 7–14.
- [3] N. Gilberg and K. G. Troitzsch, *Simulation for the social scientist*, 2nd ed. Open University Press, 2005.
- [4] A. Drogoul, D. Vanbergue, and T. Meurisse, “Multi-agent based simulation: Where are the agents?” in *MABS 2002*, 2002, pp. 1–15.
- [5] M. Richiardi, R. Leombruni, N. Saam, and M. Sonnessa, “A common protocol for agent-based social simulation,” *Journal of Artificial Societies and Social Simulation*, vol. 9, no. 1, 2006.
- [6] A. M. Law, *Simulation Modeling and Analysis*, 4th ed. McGraw-Hill, 2007.
- [7] O. Balci, “Validation, verification and testing techniques throughout the life cycle of a simulation study,” *Annals of Operations Research*, vol. 53, pp. 121–173, 1994.
- [8] F. Klügl, “Towards a formal framework for multi-agent simulation models,” Institut für Informatik, Universität Würzburg, Tech. Rep. 412, 2007.
- [9] B. Henderson-Sellers and P. Giorgini, Eds., *Agent-Oriented Methodologies*. IDEA Group Publishing, 2005.
- [10] F. Bergenti, M.-P. Gleizes, and F. Zambonelli, Eds., *Methodologies and Software Engineering for Agent Systems: The Agent-oriented Software Engineering Handbook*, ser. Multiagent Systems, Artificial Societies and Simulated Organizations. Boston, London: Springer, 2004.
- [11] G. Weiss and R. Jakob, Eds., *Agentenorientierte Softwareentwicklung*. Springer, 2004.
- [12] Q.-N. N. Tran and G. C. Low, “Comparison of ten agent-oriented methodologies,” in *Agent-Oriented Methodologies*, B. Henderson-Sellers and P. Giorgini, Eds. IDEA Group Publishing, 2005, ch. XII, pp. 341–367.
- [13] C. Bernon, M.-P. Gleizes, and G. Picard, “Enhancing self-organising emergent systems design with simulation,” in *Engineering Societies in the Agents World VII, 7th International Workshop, ESAW 2006, Dublin, Ireland, September 6-8, 2006 Revised Selected and Invited Papers*, ser. Lecture Notes in Computer Science, G. M. P. O’Hare, A. Ricci, M. J. O’Grady, and O. Dikenelli, Eds., vol. 4457. Springer, 2007, pp. 284–299.
- [14] A. Dornhaus, F. Klügl, C. Oechslein, F. Puppe, and L. Chittka, “Benefits of recruitment in honey bees: effects of ecology and colony size in an individual-based model,” *Behavioral Ecology*, vol. 17, no. 3, pp. 334–344, 2006.
- [15] V. Grimm and S. F. Railsback, *Individual-Based Modeling and Ecology*. Princeton University Press, 2005.
- [16] B. Edmonds and S. Moss, “From kiss to kids - an ‘anti-simplistic’ modelling approach,” in *Multi-Agent Based Simulation*, ser. LNAI, P. D. et al., Ed., no. 3415. Springer, 2004, pp. 130–144.
- [17] A. Pyka and G. Fagiolo, “Agent-based modelling: A methodology for neo-schumpeterian economics,” Universität Augsburg, Volkswirtschaftliche Diskussionsreihe, Tech. Rep. 272, 2005.
- [18] C. Triebig and F. Klügl, “Elements of a documentation framework for agent-based simulation models,” *Cybernetics and Men*, accepted 2009.
- [19] J. W. Haefner, *Modeling Biological Systems – Principles and Applications*, 2nd ed. New York: Springer, 2005.
- [20] J. D. Sterman, *Business Dynamics: Systems Thinking and Modeling for a Complex World*. Boston: McGraw Hill, 2000.