A negotiation approach to support interoperability in a collaborative manufacturing environment

Adina Cretan
Computer Science Department
"Nicolae Titulescu" University of Bucharest
Bucharest, Romania
E-mail: badina20@yahoo.com

Carlos Coutinho
Caixa Mágica Software/ Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR-IUL,
Lisboa, Portugal
Email:carlos.coutinho.phd@gmail.com

Ben Bratu
Atos, Big Data and Security R&D
Les Clayes-sous-Bois, France
E-mail: ben.bratu@atos.net

Ricardo Jardim-Goncalves
CTS, Departamento de Engenharia Electrotecnica
FCTUNL, UNINOVA, Caparica, Portugal
E-mail: rg@uninova.pt

***Abstract*—Maintaining the interoperability in a dynamic competitive manufacturing environment in which different business enterprises collaborate is difficult to achieve. Our approach highlights negotiation as the best solution to solve interoperability problems by reaching the common decision suitable for all managers of various enterprises in the most optimized amount of time. In this context, this paper proposes a multi-agent negotiation model, able to coordinate several negotiations taking place in parallel among multiple participants. It is described the negotiation strategy for evaluating and generating offers and the protocol for sending the offers to the other agents. This model is being implemented in the H2020 C2NET project for supporting manufacturing.**

Keywords—negotiation; enterprise interoperability; collaboration; negotiation protocol

I. INTRODUCTION

The frequent business changes in a dynamic competitive manufacturing environment threaten breaking interoperability among collaborating enterprises requiring a period of adaptation in order to renew the business relationships. Usually Small and Medium Enterprises (SMEs) cannot handle these changes leading to the need to find permanent solutions to maintain the existing interoperability and to keep their business partners.

In this respect, the proposed solution consists in a multi-agent negotiation model able to manage multiple bilateral negotiations in order to tackle the interoperability problems in business-to-business interactions within a collaborative manufacturing environment.

This paper is structured as follows: Section II presents the relation to existing theories and work; Section III presents briefly the multi-agent architecture of the negotiation system, with the main goal of supporting humans in the reestablishment of collaboration among contractual partners, in the event of breaking interoperability; Section IV describes the negotiation protocol; Section V presents the implementation of coordination mechanisms; Section VI provides a case study in the manufacturing area, within the European research project H2020 C2NET, based on the coordination of a set of bilateral negotiations. Finally, Section VII presents the conclusions on the proposed solution.

II. RELATION TO EXISTING THEORIES AND WORK

Automated negotiations have been the subject of many research papers. In this respect, Fujita [1] proposes automated agents that can estimate the opponents' strategies based on the past negotiations. Caillere et al. [2] develop a protocol and rules which help the agents to coordinate their interactions and to reach an agreement.

Other negotiation research approaches tackle the issue related to the design of a negotiation environment, considering two directions: i) the first in which the intelligent agents replace humans in negotiations; and ii) the second direction in which the intelligent software agents assist human user providing a negotiation support. Considering the first direction, Lin and Kraus [3] propose a generic environment where automated agents can proficiently negotiate with human negotiators. Regarding the second direction, several research papers propose a collaborative solution based on a service oriented architecture which helps inter-organizational information processing in distributed workflows, as in [4] and [5].

Compared with the presented state of the art, where the coordination of negotiations is handled at protocol level, the proposed approach splits the negotiation process into three discrete processes: *decision-making process*, *coordination process* and *communication process* at middleware level, allowing to be integrated in any deliberative Multi-agent Systems (MAS) or directly as a support in a human interaction negotiation system.

III. NEGOTIATION SYSTEM ARCHITECTURE

In order to implement our approach concerning the division of the negotiating process into three distinct processes (i.e., decision-making process, coordination process and communication process), it has been proposed an architecture

structured in four main layers: Negotiation Manager, Negotiation Agent, Coordination Negotiation Services and Communication Middleware (Fig. 1).

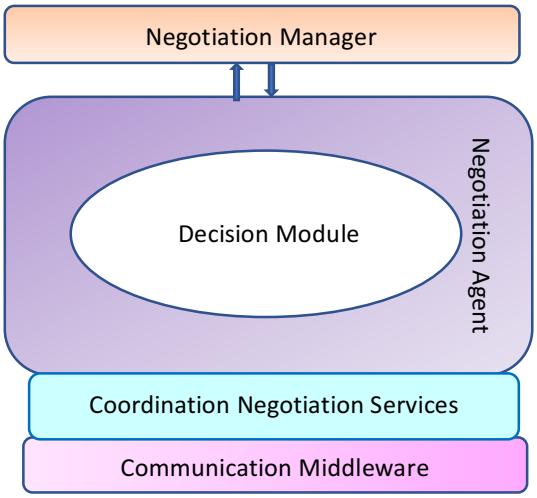

Fig. 1. The architecture of the negotiation system

A first layer, Negotiation Manager manages all business decisions regarding the creation of offers, acceptance or rejection of offers, invitation of another partner to participate in the negotiation process etc.

The Negotiation Agent, the second layer, has the role of assisting the Negotiation Manager in making decisions regarding the negotiations at a global level (i.e., negotiations with various participants on different jobs) and at a specific level (i.e., negotiations on the same job with various participants). During a negotiation, the Negotiation Agent handles one or more *Negotiation Objects*, one *Negotiation Framework,* as well as a negotiation state represented as a graph structure.

The third layer, Coordination Negotiation Services, manages the constraints of the coordination process among various concurrent negotiations.

The Communication Middleware is the fourth layer, shared by all negotiation participants ensuring, thus, the communication process.

In the proposed approach, each Coordination Service models a specific negotiation step or strategy (i.e., selection of negotiation participants; outsourcing or insourcing of a job etc.). In this respect, various Coordination Negotiation Services have been proposed [6]: *Outsrc* (resp. *Insrc*), for outsourcing (resp. insourcing) jobs by exchanging offers among partners known from the beginning of negotiation; *Block* service for assuring that a task is entirely subcontracted by the single participant; *Split* service handles the propagation of constraints among several slots, negotiated in parallel and issued from the split of a single job; *Broker* service deals with the automatic selection of possible participants in the beginning of the negotiation; *SwapIn* (resp. *SwapOut)* services implement a coordination mechanism between two ongoing negotiations to facilitate an exchange between their two tasks; *Transp* service implements a coordination mechanism between two ongoing negotiations in order to facilitate the common transport of their two tasks. These Coordination Services are able to evaluate the received offers checking whether these are valid and, further, able to reply with new offers constructed based on their particular coordination constraints. On this level, the interoperability is sustained by developing a generic coordination framework for the negotiation participants.

The advantages of the proposed negotiation architecture consist of:

- allowing a precise identification of the coordination objects;
- managing the dependencies among the existing negotiations within the manufacturing environment;
- ensuring the coordination of concurrent negotiations at the Negotiation Services level.

IV. DESCRIPTION OF NEGOTIATION PROTOCOL

The communication process is provided by Middleware layer that defines generic broadcast and synchronization mechanisms for different offers exchanged during a negotiation process. This layer is an extension of Coordination Language Facility (CLF) Middleware [7] able to support various collaborative activities taking place within the manufacturing environment [8]. This extension is called Xplore. At the Xplore Middleware level, a negotiation process is represented as a bicolored graph. The evolution of a negotiation in terms of proposals and counter-proposals is modelled by a black and white graph in which white nodes, representing *negotiation contexts*, and black nodes, representing *decision points* with multiple alternatives. Each context (white) node contains a parameter and a set of attributes with associated values. Parameter is the task to be negotiated (*Negotiation Object*) which is described in a time moment by a set of attributes that have to be negotiated depending on the specific information about the state of the negotiation in that node.

The concept of choice introduced by black nodes imposes a restriction on the construction of the graph Xplore such as: the sub-graphs that have a common black root must not have any other node in common. Therefore, the negotiation described in Fig. 2, involving an initiator (Participant P1) and two potentials partners (P2 and P3) will require an instance of a Outsrc service and two instances of the Insrc service for each possible partner.

As shown in Fig. 2, the participant P1 via its *Outsrc* service, has a complete image on the negotiation graph, while the other two participants through their *Insrc* services, have only a partial image on the negotiation graph corresponding to their own negotiations, having no information about the existence of the other participant in the negotiation process [9].

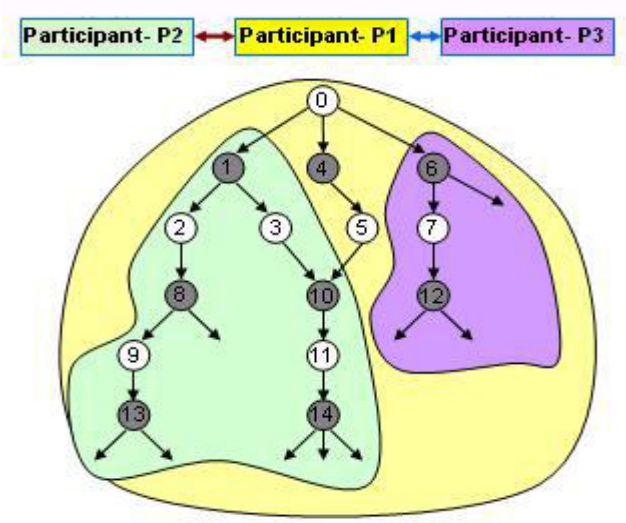

Fig. 2. Example of Negotiation Graph

Following the proposed approach, Fig. 3 provides an example of graph Xplore, in which the unique parameter of the negotiation is a printing *job* along with several attributes and ranges of possible values.

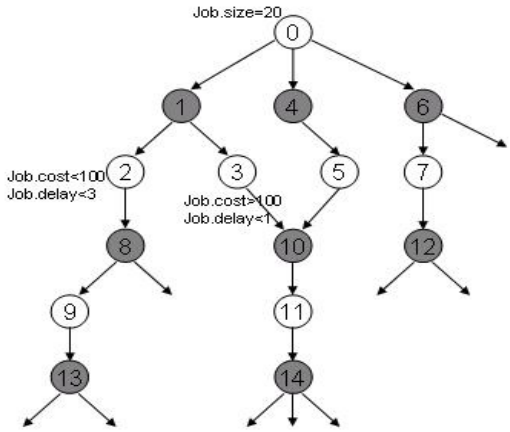

Fig. 3. The Negotiation Graph with the parameter *job*

The negotiation partners can create new branches in the Xplore graph corresponding to different solutions in terms of negotiated attributes (e.g., cost under 100 Euros or over 100 Euros). In this situation, the participants can continue the negotiation in every branch of the graph indicating, for example, another attribute *delay* (e.g., price over 100 Euros, but in a shorter time - delay of 1 day or price less 100 Euros, but in a longer time - delay of 3 days). The interaction indicating the delay (1 day or 3 days) occurs in two states of negotiation, corresponding to the two branches of the graph created by the interaction that takes place on the subject of price (e.g., the node 2 is represented by the parameter *Job* and the attributes *cost*<100 and *delay*<3).

As a consequence of the proposed approach whereby the negotiation process is a distributed process among the involved participants, the middleware Xplore models it in the same manner as a construction of a negotiation graph. Each negotiation partner has his own copy (partial) of the Xplore graph making decisions and acting only on that copy. In this respect, the goal of middleware Xplore is to maintain, for each participant, a graphical image of negotiations and synchronize the image with the partners involved in the negotiation process. In order to model this synchronization, the middleware uses six operations which are called *verbs* of the Xplore protocol. These verbs are:

- **Connect*(n, m)***: informs a coordination service (e.g., *Inscr* service) that is involved in a negotiation having the root *n* and the task to be negotiated will be identified by parameter *m*. This identification is necessary to make clearly the distinction among different graphs Xplore and the negotiated tasks for each service involved in the same negotiation.

Fig. 4 shows how the negotiation begins: the participant P1 invites in the negotiation process the two participants, P2 and P3. By using the verb *Connect* in two different nodes, the participants P2 and P3 are introduced by P1 in the current negotiation, and, depending on their root nodes they will have different images for the same negotiation.

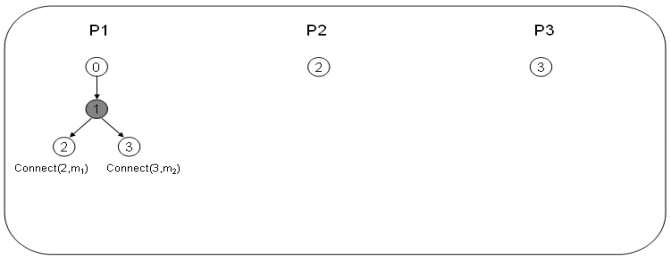

Fig. 4. Connect verb

- **Open*(n,n1,…, np)***: creates a new node *n* in the graph Xplore, with the parents nodes *n1, ..., np*. All parents nodes *n1, ..., np* (if any) must be the same color, while node *n* will be the opposite color:
  - if *n* is black, then *p* must be at least equal to 1, where *n* is a negotiation decision taken in a negotiation state resulting from the merger of the negotiation states represented by white parents nodes for p> 1; if p = 1 then the negotiation stage in which the decision has been taken is represented by one white parent.
  - if *n* is white, then *p* must be at most equal to 1, where *n* is a state of negotiation that is an alternative to the decision represented by a single parent black if p = 1 or a first state (null) of negotiation if p = 0.
- **Assert*(n, v, a, t):*** expresses the decision taken in the negotiation state represented by node *n*, such as the value of the variable *v* to have the property expressed by the term *t* on the aspect *a*.

For example, in Fig. 5, assuming that the participant P1 wants to negotiate a task with size= 20K and chooses to make the same proposal to the participants P2 and P3.

Considering that node 0 is the root of the graph, the participant P1 uses the verb *assert (0, Job, size = 20K)* to introduce in the root of the negotiation graph the attribute *size* with value = 20k.

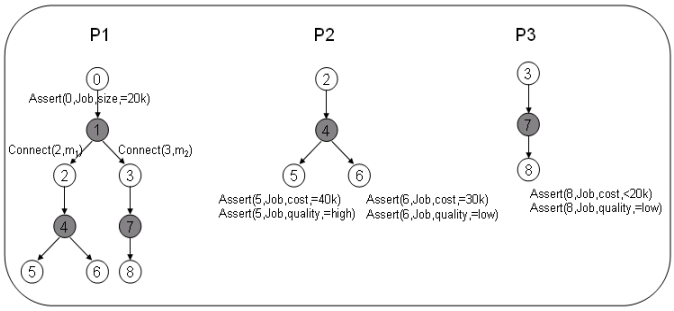

Fig. 5. Assert verb

Further, assuming that the participant P2 wants to make two separate proposals by opening, in the first stage, a black node - *open (4,2)* - and then, starting from this black node, he makes the proposals opening white nodes - *open(5,4)* and *open(6,4)* - in which he specifies the proposals - *assert(5,Job,cost,=40k) assert(5,Job,quality,=high)* and *assert(6,Job,cost,=30k) assert(6,Job,quality,=low)*.

Using the same verbs, the participant P3 makes a proposal, as well. Further, the middleware that manages communication synchronizes the operations made by the two invited participants P2 and P3 in order to allow the participant P1 to see their three proposals.

- **Request*(n, v, a)*:** in order to continue the negotiation, a participant has to notify the other partners, in the white node *n*, about the fact that he expects the assertion on the aspect *a* for the decision variable *v*.

In Fig. 6, the participant P1 announces the other two participants he expects a proposal for the attribute *delay*; This announcement is made by the verb *request(0,Job,delay)*.

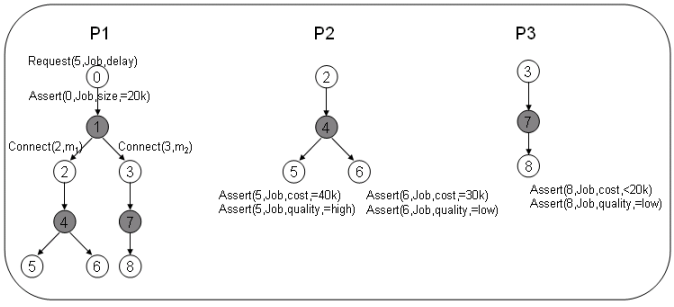

Fig. 6. Request verb

- **Ready*(n)*:** expresses that a participant has enough information in the white node *n* and is ready to accept the proposal.
- **Quit*(n)*:** expresses that a participant does not want to continue the negotiation in the white node *n*.

Assuming that the participant P2 replies with an offer completed by a value for the attribute *delay* (Fig. 7) and the participant P1 is satisfied by this proposal. In this case, P1 can stop the negotiation in the nodes 6 and 8 - *quit (6)*, and *quit (8)* - and accept the proposal of the node 5 - *ready (5)*.

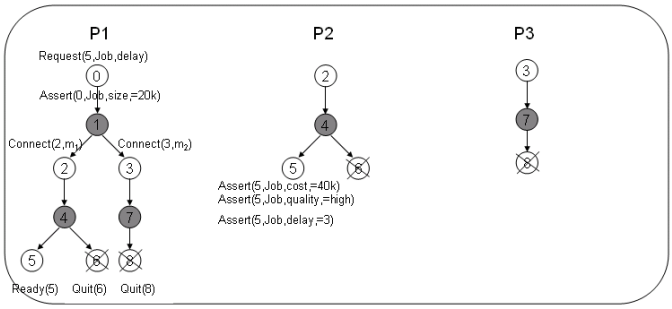

Fig. 7. Ready and quit verbs

Concluding all these aspects, we can say that the CONNECT verb allows to dynamically involve a new participant in a negotiation. The OPEN and ASSERT verbs allow a participant to build the negotiation graph, by creating and populating context nodes with information about the negotiation state at these nodes. The REQUEST verb avoids DeadLock situations by allowing participants to express their information needs on some given terms of the negotiation in order to proceed in the negotiation. The READY and the QUIT verbs allow a participant to declare respectively, that he is "ready to sign a contract" in the state of a given negotiation context, or, on the contrary, that he wants to give up the negotiation at that state (but he may pursue the negotiation in other branches).

Hence, the Middleware layer provides several generic coordination and communication mechanisms for implementation of various activities in a distributed and open environment.

Particularly, for the negotiation process, the Middleware layer supports a multi-attribute and multi-participant negotiation. In addition, the Middleware can manage in parallel and asynchronously, the various states that compose the same negotiation.

Thereby, the main feature of this Middleware is the generic proposed approach. In counterpart, this generic approach leads to the fact that data incorporated in the nodes are provided by higher levels (i.e., Negotiation Agent and Coordination Negotiation Services).

V. THE IMPLEMENTATION OF THE COORDINATION MECHANISMS

The proposed implementation is structured to allow the identification of the user part (Negotiation Manager), as well as the semi-automated part of the negotiation process which the infrastructure provided to the user.

A. Implementation Constraints

The proposed software architecture has been designed to satisfy the following implementation constraints [10]:

- The system has to support information sharing and collaborative decision-making, given the existence of several autonomous organizations which may be geographically spread. In this case, it can be expected

that a central server may not be able to support running the application for a large number of participants. Therefore, we should provide cloned coordination services, installed on several Web servers.

- The proposed coordination negotiation services should be capable of simultaneously running several negotiations and several transactions in order to provide a flexible solution to users who want to negotiate part of their tasks.
- The format of data exchanged during the negotiation should describe the negotiation object, the attributes of the negotiation task and the values of different alternative negotiation states proposed during a negotiation.

B. *The Description of Software Architecture*

The proposed software architecture described in Fig. 8 is a client-server architecture:

- On the client side, there are different manufacturing companies (e.g., Print-shops) called *Components*. A company (*component*) may choose to negotiate in a desired coordination service, as well as in a particular invocation of this service. A service invocation will create a particular negotiation graph where the company will negotiate. To view and act on such negotiation graph, we created a graphic interface called **Editor**;
- The server side includes the coordination services (e.g., *Outsrc*, *Insrc*, *Block*). The various instances of the services will be managed in parallel. For each coordination service, we provide a data structure capable of registering all instances of the respective service. This enables the server to provide various copies of existing instances to the registered services.

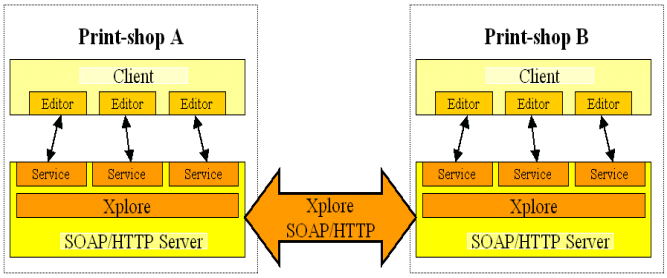

Fig. 8. The client-server architecture

Every company (print-shop) is implemented as a client application that communicates with the server using Remote Procedure Calls (RPC) where the client sends a SOAP (Simple Object Access Protocol) request to the server and the server immediately sends a SOAP reply to the client. We chose to use the SOAP protocol as it provides a simple and reliable mechanism that enables the transfer of information in a decentralized and distributed environment.

The coordination services can be implemented in the Java program allowing a distributed implementation, with the clients communicating via RPC.

The proposed **Editor** interface enables users to initiate an instance of the *Outsrc* service and then, depending on the desired negotiation tactic, to attach instances of other services. For example, if the Manager of a manufacturing company (e.g., print-shop) wants the outsourcing task to be executed entirely by a single participant, it can do so by using the *Block* service. The *Outsrc* service invites the Block service to the ongoing negotiation; then the Block service invites the *Insrc* services of each partner to negotiate with them. The coordination services manage the graph-structured negotiation.

Every instance of a service, newly invited to the negotiation, builds its own graph representing its view of the negotiation in which it is involved. Therefore, a new proposal can be initiated in the graph of an instance by adding new nodes, stating the attributes and the related values. This change in one graph is sent to all instances of the services involved in the same negotiation.

Every company is implemented as a client application and the proposed negotiation infrastructure (Negotiation Agent-Coordination Services-Middleware) is developed on a SOAP server. We made this choice considering that:

- The client application has to be used exclusively for viewing a negotiation;
- At a given moment, we have only one negotiation graph at the client side, while the server manages all negotiation graphs existing at a certain point in time.

Implementing a distributed system requires that every print-shop feature the same software structure. The protocol used in the inter-company communication is XPLORE protocol implementation.

In the next sub-sections, we will describe every part of the architecture (Editor, Negotiation Agent, Coordination Negotiation Services and Xplore), as well as the way these implementations utilize the properties of the SOAP protocol.

C. *Editor Interface*

- The proposed *Editor* is used to control the negotiation of a participant, enabling the latter to act on the negotiation graphs. Therefore, starting from this interface, a company Manager (within the developed architecture) can choose to negotiate in a particular Coordination Service and this service will create a graph instance (*Invocation*) for every new negotiation. The *Editor* represents the image of the Negotiation Agent over all its Manager's negotiations. The Manager can use the *Editor* to act on a single negotiation graph, while also being able to navigate among existing negotiation graphs and choose any of them to continue a negotiation.

In this context, Fig. 9 describes:

- In the left-hand panel: name of the company (*A0*), name of instance of ongoing service (*Out1_0*) and *Outsrc* and *Insrc* services with existing instances;
- In the middle panel: the bicolored graph representing the ongoing negotiation status viewed in the selected service instance;

- In the right-hand panel: three windows for viewing various data attached in a white node (instance attributes - **asserted**, requested attributes – **requested** and connections starting from the selected node - **connected**).

The middle panel of the Fig. 9 is a graphic implementation featuring the specific actions of a *viewer* (node selection, changes in the location and geometric size of one or several nodes etc.) and of a *controller* that enables changes in the graph structure using the Xplore protocol (the negotiation graph structure is modeled by Xplore verbs – *open, assert, request, quit, accept, connect*).

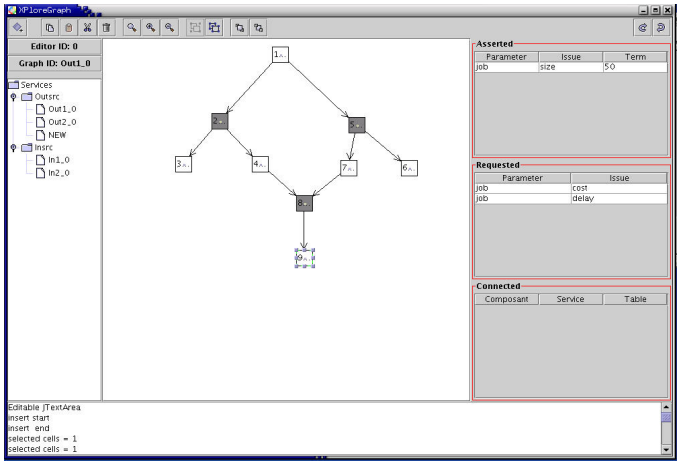

Fig. 9 - Editor interface

Moreover, we enhanced the client application by implementing several interfaces for creating and handling various Negotiation Frameworks or Negotiation Objects that a Manager can specify and also for building the Database of the collaborative partners.

In this respect, we specified and implemented two interfaces for each structure:

- one for viewing the set of all structures of the same type (for instance, Fig. 10 depicts the whole set of negotiation objects);

Fig. 10 – Interface for viewing the set of negotiation objects

- another one to create or change the fields for the respective structure type (for example, Fig. 11 is a snapshot of creating a new negotiation object).

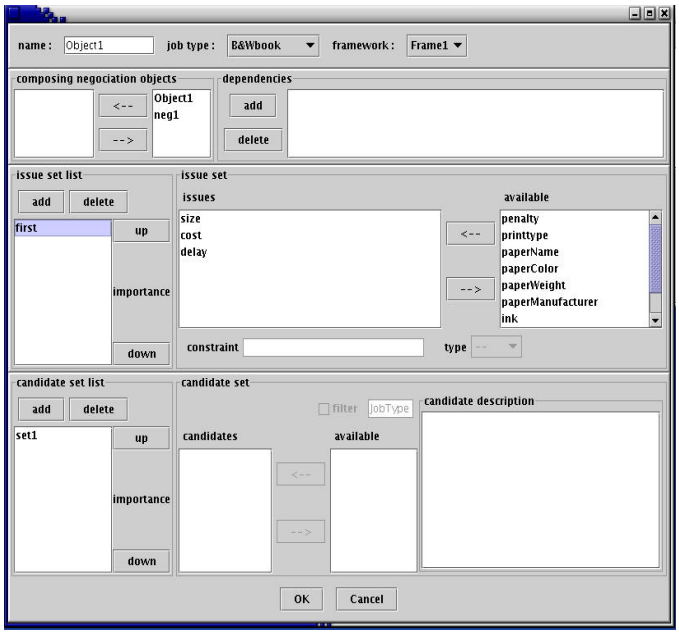

Fig. 11 – Interface for creating a new negotiation object

As described above, the application called *Editor* plays the role of interface between the Manager and the Coordination Services proposed by the infrastructure and implemented in the server. These interfaces enable the users (Managers) to join in the interaction in order to enter their conditions at the Negotiation Object and Negotiation Framework level and to make proposals in a certain negotiation.

Consequently, the Editor (as shown in Fig.12) is implementing a **Graph Representation Module** to handle a negotiation graph and a **Communication Module** to enable the interaction with the Coordination Services on the Server side through the various methods proposed by SOAP-Services.

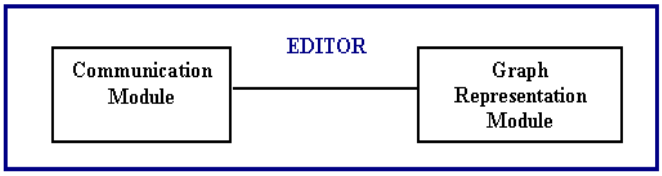

Fig. 12. Editor architecture

D. Graph Representation Module

The **Graph Representation Module** manages one or several *Graph View* (one for each conversation), such that:

- A conversation is represented by a *Negotiation Graph*;
- A *Negotiation Graph* is an oriented bicolored graph that models the topological structure of the negotiation graph presented in the conceptual design. The *Negotiation Graph* is implemented in the Editor using the *Jgraph* library. Also, all Xplore verbs - *open*

(n,n1,…,np), assert (n,v,a,t), request (n,v,a), quit (n), ready (n), connect (n,m) – are available for building the negotiation graph.

The Graph Representation Module features three parts: **Model, View** and **Controller**.

The **Model** is made of several classes that implement the *Negotiation Graph* structure and the negotiation mechanisms on the bicolored graph.

Thus, a negotiation graph is modeled through a *GraphModel* object containing several instances of the *GraphCell* object for modeling nodes and arcs (a *GraphCell* instance is a node by default). The model also contains information on the hierarchical relationship among the nodes of a graph (i.e., parent-child relationship graph). To find this information, several methods are suggested:

- *getSource* and *getTarget*, return the source node and the target node for an arc. An arc is also the child of the source node, which allows it to have several arcs stemming from the same node.
- *getChild, getChildCount, getIndexOfChild* and *getParent* methods. Parentless objects are roots and can be found using the *getRootAt* and *getRootCount* methods.

The **View** controls the layout of the negotiation graph geometric model representation and also updates and displays the graphic context that links the model (**Model**) to the template (**View**).

GraphView-type objects and *CellView*-type objects make up the image of a graph. *CellView* instances are equivalent to the *GraphCell* graph cells of the **Model**. The *GraphView* object manages the set of cells (one for each node or arc).

The **Controller** manages the rendering process by specifying the stages of cell editing and handling, as well as the other "look-and-feel" actions, such as node selection or movement. The interface is made of:

- a top-level menu enabling the user to send (Publish button) and receive (Update button) the changes for a negotiation graph;
- a central panel for viewing the negotiation graph. From this panel, the authorized actions on the negotiation graph are performed via a pop-up menu that is enabled only when a node is subject to an onmouseover event;
- a left panel where existing asserts/requests are viewed and new assert/request for a selected node are edited.

E. Communication Module

Every **Editor** features a **Communication Module** for the data transfer to and from the server**.** The Communication Module is implemented via two classes *CommunicationModule.java* and *SoapServiceCall.java.*

The *CommunicationModule* object provides methods applied by the **Graph Representation Module** to send and receive various SOAP-services data, as well as to invoke the methods proposed by these services.

The *CommunicationModule* contains:

// the object used to invoke SOAP services

private SoapServiceCall soap;

// the object used to store data received from the server

private Node updateNode;

// thread for new data monitoring

private CommunicationHandlerThread comThread;

Methods used to invoke SOAP services:

register() // is the first method invoked to get the id for the current editor;

getServices() // asks what are the existing Xplore services on the server. This method returns a string whose tokens match the names of various Xplore services. By default, this method returns the names of the *Outsrc and Insrc* services as the only existing Xplore services;

getInvocations(String serviceName) // asks what are the existing graphs for the service described by serviceName, receives a list of idGraph (String);

getGraph(String idGraph) // requests the graph description named idGraph. This method returns the graph description starting with the root node;

addData(String id_Graph, Node node) // sends the user's changes, sent data structure is a Node type structure. The sent node is the highest node in the hierarchy of the graph with the changed data;

isNewData() // intercepts the new data on the server, the return is blocked until new data arrives on the server;

getData(String id_Graph) // gets new data from the server. Data returned structure is a Node-type structure.

To enable SOAP services invocation, we implemented a *SoapServiceCall* object that hides SOAP calls. We defined two methods for calling a SOAP service method:

void init(String serviceName) // Initializes SOAP Service Facade, encoding style (SOAP envelope) and URI SOAP service;

Object invoke(String methodName, Vector params)// Remote Method Invocation asks for the method name to be invoked and for the list of parameters to be sent and returns the object received from the server.

F. Negotiation Agent – Coordination Services – Middleware

In line with the architectural design, we implemented the Negotiation Agent, the Coordination Services and Middleware Xplore in the server application.

According to the conceptual design of the negotiation process, a negotiation is modeled as a collaborative construction of a negotiation graph by the negotiation participants.

Each participant is represented by one or several coordination services that seek to synchronize the construction of the graph using the Xplore primitives. To do this, we implemented a data structure describing the negotiation graph handled by an instance of a coordination service.

We also made this instance communicate with the Editor, on the one hand, and, on the other hand, with the other services that can be located either on the same server or on other servers. Communication is encapsulated in invocations of the methods proposed by SOAP-services.

Further on, we will present the structure of data used to describe a negotiation graph, then the implementation of SOAP services.

G. Data Structures

A manufacturing company can be involved in several negotiations at the same time. Depending on how many invocations of services are involved in the negotiation process, each negotiation is supported as a set of graph data structures. Every graph is a structure of black and white nodes and data attached to these nodes. In this respect, we suggested two data structures: *Graph. java* and *Node. java*.

Graph.java-type objects contain the description of an XPLORE graph. It is composed of an instance of the *Node* object representing the graph root and a set of *Node* objects that represent the graph nodes in a parent-child relationship.
This graph-type representation is the image that an instance of a service manages at a given time. This also implies synchronization with the other instances of the services involved in the current negotiation.
To synchronise the instances involved in the same negotiation, every *Graph* object also contains the set of identifiers of the other instances managing the same negotiation graph.

The structure of the Graph object is as follows:

```
public class Graph
{
    private Node root;
    private int nodesNumber;
    private String id; // graph name is an unique identifier
    private Node[] nodesList;  // list of all nodes making up the current graph;
    private String[] relatedGraphs; // names of graphs with the same node structure, but not necessary with the same data in the nodes.
}
```

A node structure contains data identifying the node's structural features (node identifier, graph identifier etc.) and also the features of the Xplore nodes (color, asserted data etc.):

```
String idNode;
String idGraph;
String idClient;
String color;
private final int Max_Related 20;
String parents[Max_Related];
Node offsprings[Max_Related];
private final int Max_Data 50;
String assertList[Max_Data];
String requestList[Max_Data];
```

These data structures will be used by SOAP-services in the coordination and synchronization of the communication among instances of services involved in different negotiations.

H. SOAP Services

The proposed SOAP services provide particular methods for processing sets of invocations of accessible methods that provide support for the communication and synchronization between an instance of a coordination service and a client (Editor) or between different instances. In this respect, we have proposed three SOAP services represented by the following classes: *HandlerService*, *NotificationService* and *DataService*. We will further detail *DataService* only, as it constitutes the SOAP service where the entire negotiation mechanism is implemented (Negotiation Agent-Coordination Services-Middleware). The other two SOAP services being exclusively used to maintain a synchronization between the image of a negotiation graph in the client (Editor) and the one controlled by the coordination services in the server.

At the SOAP server level, the number of graphs depends on the number of clients registered on the server, namely the number of Editors that a manufacturing company uses, the number of negotiations that the company attends and eventually the number of instances of the coordination services involved in the concerned negotiations. The data structures used to record this information are as follows:

Client structure

```
public class Client{ // an instance of this object is created for every service registered on the server.
String idClient;
LinkedList outsrc; // list of Xplore Outsrc's invocations accessible to this client;
LinkedList insrc; // list of Xplore Insrc's invocations accessible to this client;
```

`LinkedList bloc;` // list of Xplore Block's invocations accessible to this client;

`LinkedList broker;` // list of Xplore Broker's invocations accessible to this client;

`LinkedList split;` // list of Xplore Split's invocations accessible to this client;

`LinkedList swapOut;` //list of Xplore SwapOut's invocations accessible to this client;

`LinkedList swapIn;` //list of Xplore SwapIn's invocations accessible to this client;

`LinkedList transp;` //list of Xplore Transp's invocations accessible to this client;

```
Boolean newData;
                    }
```

- *Map of invocations*

`public Hashtable mapGraphs = new Hashtable();` // this hashtable contains all the invocations created by the services registered to this server. The access key for an invocation is the id of the graph structure created during a Xplore service invocation.

- *Vector of clients*

`public Vector vectorClient = new Vector(20,10);` // this vector contains all registered clients.

- *Map of online Graphs*

`public Hashtable mapInvocations = new Hashtable()` // this hashtable contains all the invocations currently used by clients. The access key is the name of the invocation and the returned data is the client id.

SOAP DataService proposes several types of methods to enable the client-server communication, the handling of negotiation graph-type structures and the coordination and synchronization mechanisms provided by the coordination services and the Xplore protocol.

Fig. 13 shows the invocations of the methods proposed by DataService for client registration:

`register()` // this is the first method a client invokes to get a unique Id,

`deregister(Id_Client)` // deletes the client Id from the server list.

`getServices()` // gets different coordination services that are proposed (e.g., Outsrc, Insrc etc.). This method returns a string whose tokens correspond to the name of the various Xplore services.

`getInvocations(String id_Client, String serviceName)` // invokes the creation of a service instance. This method returns a string representing the Xplore services instance id.

`getGraph(String id_Client, String id_Graph)` // gets the specified graph.

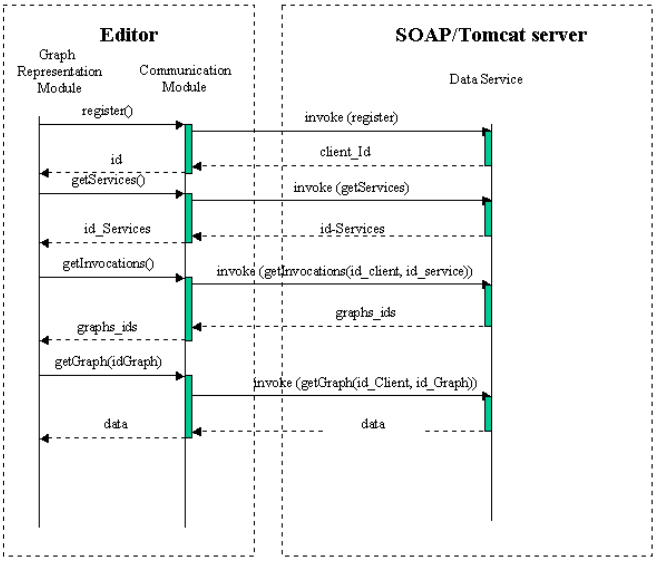

Fig. 13 – Method invocations in DataService

As specified in the architecture design, instances of services involved in a negotiation communicate exclusively via Xplore verbs.

Therefore, DataService also provides the methods to facilitate communication among instances on the same server, but also among instances on different servers:

```
open (String idGraph, String idNode, String idsParents);
assert(String idGraph, String idNode, String param, String issue, String term);
request(String idGraph, String idNode, String param, String issue);
quit(String idGraph, String idNode);
ready(String idGraph, String idNode);
connect(String idGraph, String idNode, String idService, String map);
```

The communication among instances of the coordination services relies on Xplore protocol features. Thus, the interactions are implemented as RPC (Remote Procedure Call) invocations that carry the signature of various Xplore verbs.

The following example presents a RPC invocation that uses the SOAP message format representing the description of

the Xplore verb *assert* by using the XML scheme (only the Body part of the SOAP message envelope is shown):

The *assert* verb description is:

```
<element name= "assert"
      base="tns:assert"/ >
<complexType name="assert">
  <element name= "node" type="tns:node"/>
  <element name="parameter"
      type="tns:parameter"/>
  <element name="issue"
      type="tns:issue"/>
  <element name="term" type="tns:term"/>
</complexType>
```

The method is described as follows:

```
<SOAP-ENV:Body>
  <m:assert  xmlns:m="some-URI">
    <node>id_node</node>
    <parameter>parameter_name</parameter>
    <issue>issue_name</issue>
    <term>term_name</term>
  </m:assert>
</SOAP-ENV:Body>
```

The proposed implementation of the negotiation process comprising the description of negotiation objects, the mechanisms for sending proposals and counterproposals, the coordination modules and the synchronization mechanisms. This implementation adds various pieces for building complex multi-agent conversations. Integrating interoperability technologies, such as HTTP, SOAP, XML/XSL and Servlet, allows for smoothly attaching agents that carry conversations with other agents or human users on the Web.

VI. CASE STUDY FROM H2020 C2NET

The Horizon 2020 C2NET project envisages the support of manufacturing companies like the French dermo-cosmetics factory Pierre Fabre with a cloud-based environment that fosters the interoperability of the factory with its supply chain (suppliers, partners, customers). The project features the possibility of analysing and optimising manufacturing plans involving several partners who can choose to negotiate any change that may occur in their collaborative environment which can lead to breaking interoperability. The proposed solution provides support for reaching an agreement in the shortest possible time in an automatic and autonomous manner with little support of human interaction.

The project is implementing the proposed negotiation model in order to capture all the negotiation steps and decisions, so that at any time it is possible to roll back to one or more stages of the negotiation to retake the environment to alternative decisions or even to "what-if" scenario analysis.

VII. FINAL CONSIDERATIONS

This paper describes the implementation of the negotiation coordination model via a three-layered architecture: *Negotiation Agent, Coordination Negotiation Services* and *Middleware* Xplore.

This structure is in line with our approach of splitting the negotiation process into three discrete processes: *decision-making process, coordination process* and *communication process*.

The communication process is managed by the middleware layer that defines the generic mechanisms of communication and synchronization among several agents. At the middleware level, communication is based on the Xplore protocol that enables the management of the concurrent negotiations where, at any moment, participants can choose to simultaneously negotiate in several negotiation states.

The coordination process is managed by the coordination negotiation services layer.

The main feature of this approach is the fact that the coordination process is fully distributed on several coordination modules allowing to be defined several specialized services that can be used in any negotiation. This distribution of coordination constraints also allowed the services to run simultaneously, which enhanced the efficiency of the system, making it capable of evaluating several negotiation offers at the same time.

The decision-making process is provided by the Negotiation Agent layer that models the support mechanisms for the interaction processes within the collaborative manufacturing environment, mainly, for creating offers and making decisions in a negotiation. This layer manages the decisions that can be made on the negotiation strategy for evaluating and generating offers and on the protocol for sending the offers to the other agents. The goal at this level is to allow the human user to intervene in the decision making process. We can thus separate the decision making process from agents, which reinforces the generic applicability of our negotiation model.

ACKNOWLEDGMENT

The authors wish to acknowledge the support of the European Commission through the funding of the H2020 C2NET project for their support, interaction and contribution in the development of the business case that is presented on this paper.

REFERENCES

[1] K. Fujita, "Automated Negotiating Agent with Strategy Adaptation for Multi-times Negotiations", chapter in *Recent Advances in Agent-based Complex Automated Negotiation*, Studies in Computational Intelligence, Vol. 638, pp 21-37, 2016.

[2] R. Caillere, S. Arib, S. Aknine, and C. Berdier, "A Multiagent Multilateral Negotiation Protocol for Joint Decision-Making", chapter in *Recent Advances in Agent-based Complex Automated Negotiation*, Studies in Computational Intelligence, Vol. 638, pp 71 - 88, 2016.

[3] R. Lin and S. Kraus, "Can Automated Agents Proficiently Negotiate with Humans," Communic. of the ACM, Vol. 53/1, pp. 78-88, 2010.

[4] C. Badica, S. Ilie, M. Kamermans, G. Pavlin, and M. Scafes, "Using Negotiation for Dynamic Composition of Services in Multi-

organizational Environmental Management," Environmental Software Systems. Frameworks of Environment, Vol. 359, pp 177-188, 2011.

[5] A. Penders, G. Pavlin, M. Kamermans, "A Collaborative Approach to Construction of Complex Service Oriented Systems." *Intelligent Distributed Computing IV. Studies in Computational Intelligence*, vol. 315, pp. 55–66, Springer, 2010.

[6] A. Cretan, C. Coutinho, B. Bratu, and R. Jardim-Goncalves, "NEGOSEIO: A Framework for Negotiations toward Sustainable Enterprise Interoperability," *IFAC Journal Annual Reviews in Control*, Vol. 36, Issues 2, pp. 291-299, 2012, DOI=10.1016/j.arcontrol.2012.09.010.

[7] J.-M. Andreoli, D. Arregui, F. Pacull, M. Riviere, J.Y. Vion-Dury and J. Williamowski, "CLF/Mekano: *A Framework for Building Virtual-Enterprise Applications*". In Proc. of EDOC, Mannheim, Germany, 1999.

[8] J.-M. Andreoli and S. Castellani, "Towards a Flexible Middleware Negotiation Facility for Distributed Components". In Proc. of "E-Negotiation" DEXA, Munich, Germany, 2001.

[9] C. Coutinho, A. Cretan, C.F. da Silva, P. Ghodous, and R. Jardim-Goncalves, "Service-based Negotiation for Advanced Collaboration in Enterprise Networks", Journal of Intelligent Manufacturing, Volume 27, Issue 1, pp. 201–216, February 2016, DOI=10.1007/s10845-013-0857-4.

[10] C. Coutinho, A. Cretan, and R. Jardim-Goncalves, "A Negotiation Model for Concurrent Engineering", Proceedings of the 7th International Systems & Concurrent Engineering for Space Applications Conference (SECESA 2016) pp. 1210 - 1218, October 2016, Madrid, Spain.