Producing enactable protocols in artificial agent societies

George K. Lekeas[1], Christos Kloukinas[1], and Kostas Stathis[2]

[1] City University London, Northampton Square, London EC1V 0HB,
{g.k.lekeas,c.kloukinas}-at-soi.city.ac.uk
[2] Royal Holloway, University of London, Egham Surrey TW20 0EX,
kostas.stathis-at-cs.rhul.ac.uk

Abstract. This paper draws upon our previous work [7, 16] in which we proposed the organisation of services around the concept of *artificial agent societies* and presented a framework for representing roles and protocols using LTSs. The agent would apply for a role in the society, which would result in its participation in a number of protocols. We advocated the use of the *games-based metaphor* for describing the protocols and presented a framework for assessing the admission of the agent to the society on the basis of its *competence*. In this work we look at the subsequent question: *what information should the agent receive upon entry?*. We can not provide it with the full protocol because of security and overload issues. Therefore, we choose to only provide the actions pertinent to the protocols that the role the agent applied for participates in the society. We employ *branching bisimulation* for producing a protocol equivalent to the original one with all actions not involving the role translated into silent (τ) actions. However, this approach sometimes results in *non-enactable* protocols. In this case, we need to *repair* the protocol by adding the role in question as a recipient to certain protocol messages that were causing the problems. We present three different approaches for repairing protocols, depending on the number of messages from the original protocol they modify. The modified protocol is adopted as the final one and the agent is given the role automaton that is derived from the *branching bisimulation* process.

1 Introduction

Ubiquitous computing envisages objects with information processing and communication capabilities that will assist users in their daily tasks [18]. An example of such an setting could be a "user's personal assistant" (UPA) running on a Personal Digital Assistant (PDA). It will have knowledge of the user's timetable and assist him on the task(s) he has to carry out. The UPA could, for example, determine the location of the user and if he has to be at the airport in a short period of time order him a taxi.

In this example, the context and/or location in which the UPA is deployed play a significant role, as it will have to use a local taxi service or identify the products that are of interest to the user it represents. Furthermore, such an application will need to have a number of properties such as *autonomy* and *pro-activity*. This is the case as some of the taxi services the UPA is using might temporarily be down or not accepting a certain method of payment. On the other hand, it will need to be pro-active and be able to make decisions on what services to contact and what resources to use.

A paradigm that fits these requirements is this of a single *agent* or *Multi-Agent System* [12]. Such systems exhibit autonomy, reactivity and pro-activeness within the social context they operate (social ability). Moreover, in a *Multi-Agent System* no agent has complete ability to solve the problem, data is spread across the system, no agent can control the whole system and the computation is asynchronous; UPA will need the collaboration of other agents providing the required *services*.

In [16] we proposed the organisation of services around *artificial agent societies*. These would be *semi-open* in the sense of [3] (i.e. new members are accepted only on completion of a successful application). We should note here that *openness* is considered from a *membership* viewpoint and not, for example, from an *agent communication language* or *agent architecture* perspective.

The agent will choose the society to apply for membership on the basis of its *service needs* and apply for a *role* $\mathcal{R}$ in it. It will have to submit its *communication abilities* (i.e. the set of messages that it can utter/understand). These will be judged against the requirements of the protocols that $\mathcal{R}$ is participating in. The requirement is that the agent should be able to understand the messages that it can receive, as well as be able to utter the messages that $\mathcal{R}$ can send.

The representation of protocols is done using the games-based metaphor [15], which we extended to include different representations for the state of the game. This metaphor is not related to game theory; we are simply using the notion of game to represent the evolution of a protocol and not to quantify the agent strategies (which are, in general, unknown). The representation of the game as a protocol should allow for the representation of protocol states. These are described by the values of a number of properties we are interested in; e.g. who was the last player, who is the next one and what is the last move made in the game. In [15] *destructive assignment* has been used for this purpose, i.e., every time there is a change in the value of a property the old value is deleted and the new one is inserted. However, this is not the only option. *Situation Calculus* [9] can be used to represent the state as a sequence of actions forming a *situation*. On the other hand, if we are interested in a game that has concurrent moves, *Event Calculus* [8] could be used to describe the game state as events happening at specific time points. Finally, *commitments* [17] could be used.

Assuming that the agent in question is accepted into the society, there is a new issue of *what part of the protocol it should receive*. It could, of course, be provided by the full protocol but that might not always be easy e.g. for security or information overload problems. A procedure is, thus, needed for providing the agent only with the protocol information needed. This procedure should discard (hide from the agent) any parts of the protocol there is no need to know about as it is not involved in those. It should, also, ensure that there are no *structural* problems with the protocol that the agent receives, i.e., it is enactable. This means that any time the agent needs to take a decision as to what action to perform next, all information needed for making the decision is available to it.

The rest of the paper is structured as follows: Section 2 provides a quick overview of bisimulation, whereas Section 3 describes NetBill, our working example. We present our approach for creating the role automata in Section 4. In Section 4.1 we present three

approaches for repairing non-enactable protocols . Finally, Section 5 discusses related work and we conclude the paper in Section 6.

2 Bisimulation

Bisimulation [10] is a way of minimising LTS on abstract (silent) actions while preserving the properties of the original model. It can be computed automatically without any human involvement.

Formally, it can be defined as [13]: A binary relation $\mathcal{R}$ on the states of a *Labelled Transition System* is *bisimulation* if whenever $s_1 \mathcal{R} s_2$:

$$\begin{aligned} &\text{for all } s_1' \text{with } s_1 \xrightarrow{\mu} s_1', \text{there is } s_2' \text{ such that } s_2 \xrightarrow{\mu} s_2' \text{ and } s_1' \mathcal{R} s_2' \\ &(\forall s_1'.s_1 \xrightarrow{\mu} s_1' \Rightarrow \exists s_2' \colon s_2 \xrightarrow{\mu} s_2', s_1' \mathcal{R} s_2'); \\ &\text{the converse, on the transitions emanating from } s_2 \\ &(\forall s_2'.s_2 \xrightarrow{\mu} s_2' \Rightarrow \exists s_1' \colon s_1 \xrightarrow{\mu} s_1', s_2' \mathcal{R} s_1'). \end{aligned} \tag{1}$$

As the state of the protocol can be determined at any stage by the actions that have been already executed and the choice of what action to execute next, two equivalent (*bisimilar*) systems should represent the same evolution. This means that for any evolution of the first system (the original protocol), the second system (bisimulated model) should be able to evolve in the same way and any choice of actions in the first system should exist in the second system as well.

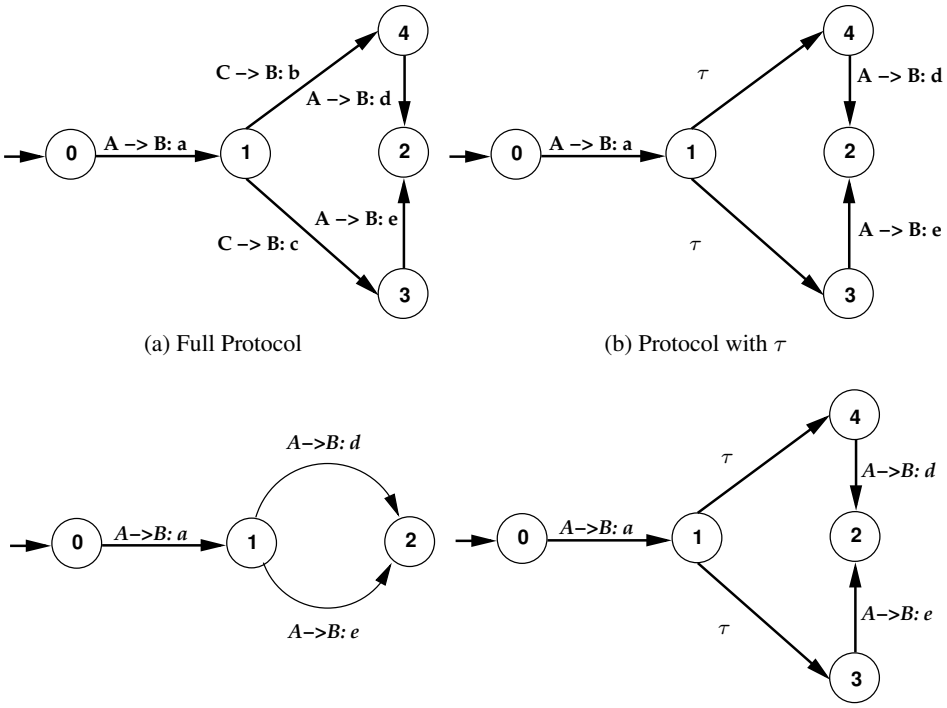
Any action the role in question is not involved in, either as a sender or amongst the recipients, is replaced by a silent (τ) action. Depending on how silent actions are treated, we distinguish between different types of bisimulation. The first option is to merge all silent actions with the first non-silent one, i.e. $\tau^\star\alpha \equiv \alpha$. This is a quick and easy way of dealing with τ actions, but it does not respect the structure of the protocol.

Branching bisimulation rectifies this by considering the structure of the LTS as well. Two LTS *P* and *Q* are branching bisimilar via a relationship *R* if: (i) their initial states are related via *R* and (ii) if r and s are related by *R* and $r \xrightarrow{\alpha} r'$, then either $\alpha = \tau$ or there exists a path $s \Rightarrow s_1 \xrightarrow{\alpha} s_2 \Rightarrow s'$ such that r and s_1, r' and s_2 as well as r' and s' are related by R.

The difference between the two types of bisimulation can be seen in Fig. 1.

Fig. 1a shows the original protocol with three roles, *A*, *B* and *C*. The protocol starts with role *A* sending message α to role *B* and afterwards role *C* has the option of sending *B* either *b* or *c*. Finally, role *A* can send role *B* either *d* or *e* but this choice is not independent of the previous steps. It depends on what message role *B* received. Fig. 1b shows the role automaton for role *A* by replacing any non-observable actions (i.e. actions that the role is not involved in as sender or recipient) with τ. The result of $\tau^\star\alpha$ bisimulation is shown in Fig. 1c. According to this, role *A* sends message α to role *B* and then it can send *B* either *d* or *e*.

However, this is not accurate. The choice of the second message is not with *A*, but depends on the choice that *C* made on the previous step. This is knowledge that role *A*



(a) Full Protocol (b) Protocol with τ

(c) Result of $\tau^{\star}\alpha$ Bisimulation (d) Result of Branching Bisimulation

Fig. 1: Non-implementable protocol due to incomplete knowledge

does not have (in effect, it does not know neither whether role *C* acted nor what message it chose to send). Branching bisimulation in Fig. 1d takes this into consideration by keeping the two branches with τ actions and not discarding them, even if they do not represent role *A*'s knowledge. This means that for role *A* the protocol that it should receive should be the same as the original one with the τ actions, even if they do not represent role's knowledge.

3 The Netbill Protocol

In this section, we introduce a variation of the e-commerce protocol NetBill [6]. This can be used by a society that aims at allowing *merchants* to sell goods to *customers* and make use of *payment gateways* in order to collect payment. An agent wishing to enter a society where *Netbill* is available will have to apply for the role of customer, merchant or gateway depending on the goal it wishes to achieve when entering the society. In the original protocol, there are three roles - *customer* (c), *merchant* (m) and *gateway* (g)- and eight overall steps for a customer to purchase goods from a merchant and the merchant to process payment for the order through NetBill's gateway. These are depicted in Fig. 2 and are as follows:

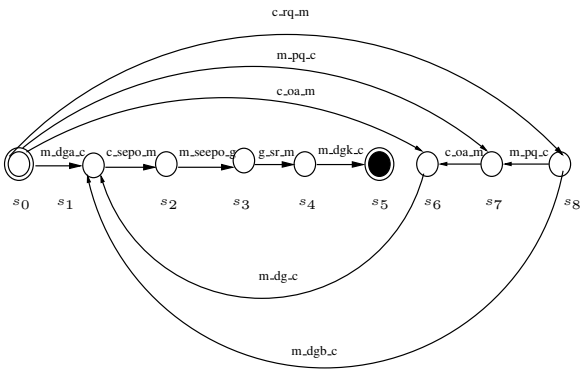


Fig. 2: A variant of the NetBill protocol

- The customer requests a quote for some digital goods from a merchant - see the transition $(s_0, (c, rq, \{m\}), s_8)$, i.e., from state 0 to state 8, labelled as $(c, rq, \{m\})$.
- The merchant provides a quote to the customer - $(s_8, (m, pq, \{c\}), s_7)$.
- The customer accepts the quote made by the merchant - $(s_7, (c, oa, \{m\}), s_6)$.
- The merchant proceeds to deliver the ordered goods encrypted with a key K - $(s_6,(m, dg, \{c\}), s_1)$.
- The customer signs an *Electronic Purchase Order* (EPO) with the merchant - $(s_1, (c, sepo, \{m\}), s_2)$.
- The merchant signs in its turn the EPO and sends it to the NetBill gateway - $(s_2, (m, ssepo, \{g\}), s_3)$.
- The NetBill gateway internally checks the information on the EPO, transfers the money and ends by sending the merchant a receipt - $(s_3,(g, sr, \{m\}),s_4)$.
- Finally, the merchant sends the customer the key needed to decrypt the goods it purchased - $(s_4, (m, dgk, \{c\}), s_5)$.

We made the following additions to the original NetBill protocol to create one with branching structure so that we can illustrate problems when the agent has to make a decision but does not have all the information required:

- The merchant can now make a price quote directly - $(s_0, (m, pq, \{c\}), s_7)$; e.g., in the case of a promotional offer.
- The merchant could select to deliver the goods as its first move - $(s_0, (m, dga, \{c\}), s_1)$; e.g., when the customer has good credit and solid reputation with that merchant. In this case, the encryption method used in the delivery can be more relaxed than the normal one as the process involves a trusted customer.
- The customer might accept the merchant's quote directly - $(s_0,(c, oa, \{m\}), s_6)$; e.g., when the merchant is trusted or this is a recurring order.
- On reception of a quote request, the merchant can make the quote and ship the goods directly without waiting for a formal acceptance of the quote - $(s_8, (m, dgb, \{c\}), s_1)$; e.g. when dealing with a trusted customer or a recurring order. The delivery and encryption method will have to be different again, as if it is a recurring order it will mean that the customer is low on stock for this particular item.

4 Producing the final enactable protocol

In order to derive the role automata for each individual role involved in the protocol, the followed process is applied:

1. prepare the initial role automaton, i.e. the automaton we get from the original protocol automaton by replacing actions for which the role is neither the sender nor amongst the recipients by τ;
2. run *branching bisimulation* on the resulting automaton;
3. examine the resulting automaton for the presence of τ actions;
   (a) if τ actions exist but not make the protocol non-enactable, this is the protocol that the role receives;
   (b) if τ actions exist and they make the protocol non-enactable, then the protocol is repaired using one of the approaches in Section 4.1 and we start over with the *updated* protocol automaton.

By following this process, the protocol for the gateway role of the NetBill protocol is reduced to two transitions and three states, as shown in Fig. 3.

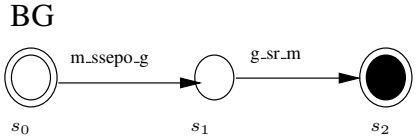


Fig. 3: The gateway role of NetBill after running branching bisimulation

The resulting protocol for the merchant agent would be the whole protocol, as the merchant is involved in all communications, while for the customer agent it would be the whole protocol except for the messages involving the gateway agent. The customer needs have no knowledge of these.

4.1 Protocol Repair

The NetBill protocol has been decomposed into role automata with no silent actions in them, as it is a well designed protocol. However, the breakdown of a protocol into its constituent roles need not always produce enactable specifications. If the resulting role automaton contains silent (τ) actions, then repair might be required. The repair process takes place at step 3b of Section 4 and consists of adding the role in question to the recipients of certain moves from the original protocol. The choice of the moves will depend on the algorithm we choose for the repair; the following sections describe three such algorithms starting with the one that will make most repairs to the protocol and finishing with the one making the least.

Updating all τ actions One approach is to find the equivalent states in the original protocol of the problematic states in the bisimulated one and add the role as a recipient to any messages originating from these states in the protocol. The algorithm is described in Listing 1.1.

```
1 // Game Protocol ⇒ GP, Role Protocol ⇒ RP
2 repair(GP, RP_badState, GP_role) {
3   GP_class= equivalence_class(RP_badstate, GP);
4   // Add role to the recipients of the moves of these states
5   foreach (GP_state in GP_class)
6     foreach (GP_tran from GP_state.transitions)
7       GP_tran.move.receivers = GP_tran_move.receivers ∪ GP_role;
8   }
```

Listing 1.1: Updating all silent transitions

This algorithm repairs the protocol by adding the extra information that was missing and was causing the occurrence of the τ move, i.e., adds the role in question to the recipients of the communication act. At the beginning, we calculate all states from the original protocol that are in the equivalence class of the originating state of the transition with the silent move in the bisimulated protocol. Once these are found, for every transition that starts from these states in the original protocol, the set of receivers is updated with the inclusion of the role whose automaton we are calculating.

Updating frontier τ actions Another approach would be to repair a few transitions of the original protocol, those that start from any state in the original protocol that belongs to the same equivalence class as the original state of the silent action in the bisimulated protocol and finish in any of the states belonging to the same equivalence class as the end state of the same transition. The intuition here is that τ transitions within states of the same equivalent class will not be present in the resulting role automaton, so no repair is needed.

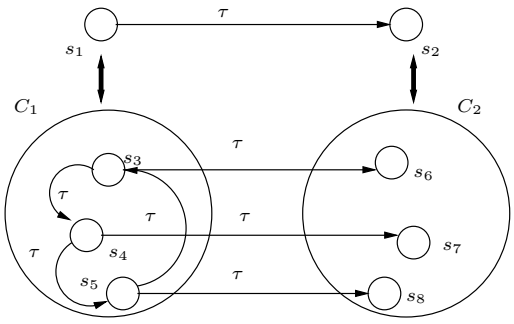


Fig. 4: Branching Bisimulation Equivalence Classes

In Fig. 4 after running branching bisimulation we have states s_1 and s_2 linked with a τ transition. However, as branching bisimulation is an equivalence relation placing states into equivalence classes, each of these two states would belong to an equivalence class of states from the original automaton. In this case, we have two equivalence classes $C_1 = \{s_3, s_4, s_5\}$ (represented by s_1) and $C_2 = \{s_6, s_7, s_8\}$ (represented by s_2). By looking at the transitions, we can see that the transitions from states belonging to class C_1 to states belonging to class C_2 are all τ transitions that need to be repaired. The benefit, however, in comparison with the approach described in Section 4.1 is that we do not repair any silent transitions *internal* to the class , i.e., the transitions from s_3 to s_4, s_4 to s_5 and s_5 to s_3.

The algorithm that performs the repair is described in Listing 1.2:

```
1  // Game Protocol ⇒ GP, Role Protocol ⇒ RP
2  repair(GP, RP_Transition, GP_role) {
3    RP_initial_state = RP_Transition.initial_state;
4    RP_end_state = RP_Transition.final_state;
5    GP_equiv_initial_states = equivalence_class(initial_state,
6  GP);
7   GP_equiv_end_states = equivalence_class(RP_end_state,GP);
8   //Add role in the recipients of the moves
9   //of those transitions that start in
10  //GP\_equiv\_initial\_states, end in GP\_equiv\_end\_states
11  //and is a silent transition in the original protocol
12  foreach (GP_tran from GP_state.transitions) {
13   GP_initial_state = GP_tran.initial_state;
14   GP_final_state = GP_tran.final_state;
15   GP_m = GP_tran.move;
16   GP_recipients = GP_tran.recipients;
17   if (GP_initial_state ∈ GP_equiv_initial_states ∧
18    GP_final_state ∈ GP_equiv_end_states ∧
19     GP_role ∉ GP_recipients)
20    GP_tran.move.recipients = GP_tran.move.recipients ∪
21                                      GP_role;
22   }
23 }
```

Listing 1.2: Updating silent actions by looking at equivalence groups

Updating selected τ actions Our approaches to protocol repair so far, have considered silent actions as something that needs to be removed from the role's final automaton - their presence would imply lack of knowledge and failure in implementation.

However, this is not always true. A role will need to have a silent action repaired only if it is causing problems in the role's action selection process. Assuming a branch where the first move in both leaves is τ, the following combinations exist for the follow-ups:

- the two actions following the silent ones are both receive actions for the role - in that case, we do not need to repair the transition as the role has no decision to make and just waits to receive a message;
- the two actions following the silent ones are both send actions for the role and they are different in terms of either the move or the recipients of the move (or both); in this case repair is needed so that the role will have the required information to decide on which move to pursue;

– one of the following moves is a send, while the second one is a receive; we need to repair the protocol in this case too, as the role in question will need the extra information to decide whether it will wait to receive the prescribed message or go ahead and send a message.

If such moves are found in a role's LTS, then they need to be repaired. This presents the overhead of having to examine a much larger section of the protocol every time we come across a silent move, but gives smaller final protocol sizes.

The algorithm for repairing a protocol in this way is shown in Listing 1.3 (this time we have to include the role LTS as well).

```
1  // Game Protocol ⇒ GP, Role Protocol ⇒ RP
2  repair(GP, RP_Transition, GP_role, RP) {
3    RP_initial_state = RP_Transition.initial_state;
4    RP_end_state = RP_Transition.final_state;
5    // check if the transition needs to be repaired
6    RP_outgoing_transitions = find_outgoing(RP_initial_state);
7    forall ( t ∈ RP_outgoing_transitions,k ∈ RP_outgoing_transitions, k ≠ t){
8          if ( t.Move == "tau" ∧ k.Move == "tau"){
9              final_state_t = t.FinalState;
10             final_state_k = k.FinalState;
11             outgoing_transitions_newt = find_outgoing(final_state_t);
12             outgoing_transitions_newk = find_outgoing(final_state_k);
13             forall (r ∈ outgoing_transitions_newt ∧ s ∈
                    outgoing_transitions_newk){
14                Move_1 = r.Move;     Move_2 = s.Move;
15                sender_1 = r.Sender; sender_2 = s.Sender;
16                Recipients_1 = r.Recipients; Recipients_2 = s.Recipients;
17                 if ((sender_1 == sender_2 == GP_Role) ∧ ((Move_1 ≠ Move_2) ∨ (
                       Recipient_1 ≠ Recipient_2))  ∨
18                (Sender_1 == GP_Role ∧Sender_2 ≠ GP_Role ∧ GP_Role ∈ Recipient_2)){
19                   // repair process
20                  initial_equiv =equivalence_class(RP_initial_state,GP);
21                  end_equiv = equivalence_class(RP_end_state,GP);
22                  forall (v ∈ GP.Transitions) {
23                    initial_state = v.InitialState;
24                    final_state = v.FinalState;
25                    if (initial_state ∈ initialequiv ∧
26                        final_state ∈ endequiv)
27                          v.Recipients = v.Recipients ∪ GP_Role;
28                  }
29                }
30         }
31     }
32 }
```

Listing 1.3: Updating selected silent transitions for role R

Example of Protocol Repair As an example of protocols requiring repair, we can look at the example in Fig. 1d. According to $\tau^\star\alpha$ bisimulation there is no need for repair as no silent actions are present in the resulting automaton. However, when running *branching bisimulation* two silent actions remain. The issue here is that role *A* arrives at a point where it has to make a decision as to which message to send to role *B*, but this decision will depend on the previous decision of role *C* for which *A* has no information about.

In this case, because of the size and the structure of the protocol, all repair algorithms will require the addition of role *A* to the recipients of messages starting from state one and emanating to states three and four. Thus, role *A* should receive all messages of the protocol and receives the protocol in Fig. 5.

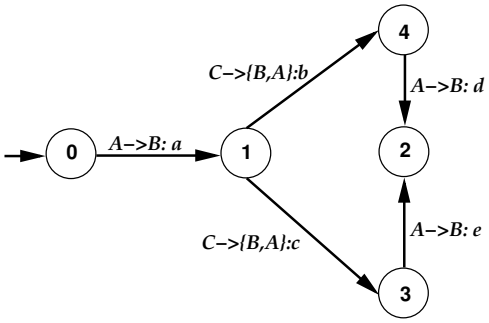


Fig. 5: Final Protocol for role A

5 Related Work

The concept of breaking down (& repairing) a protocol into constituent roles has been studied using a variety of approaches and protocol representations. In [4], Desai et al. identify the dangers of moving from the global view of a choreography (or protocol) to a local view of a single role (or agent) in either web service or multi-agent systems applications. This is important as the shift of viewpoint and the respective limitations on what the web service (or agent) can observe might mean that in the isolated agent view, there might be not enough information to implement their role specification in the choreography (or protocol).

Their description of the protocol is in a form of rules of the type $\alpha \Rightarrow \beta$. They demand that the description of the protocol always allows any proposition that is part of a rule's consequent to be part of another rule's antecedent and reachable from the beginning of the protocol. As a result of these rules, all protocols are *enactable*.

Furthermore, since they look at protocols as distributed entities and as a composition of roles, they provide an algorithm for deriving a *role skeleton*, i.e., the local view of the interaction that a role will have of the protocol including its own message exchanges. The role will need to know the messages it can send and receive, as well as any facts that enable them and lead to the creation (or discharge) of commitments (obligations of the role to bring about certain properties). The main idea in the algorithm for working out the role skeleton for a certain role is that if the role does not have knowledge of the immediate proposition needed to make a decision as to how to proceed, it should be possible to backtrack and find another one that leads with certainty to the one been examined. If the role needs to know α but it does not, then the role should go back in history and find β so that the role knows it and $\beta \rightarrow \alpha$. This algorithm works on the assumption that protocols are *enactable*. However, if they are not, there is no proposed action to rectify the problem.

Bouaziz [2] uses XML and XSD schema to describe a protocol ontology and views role as a component that can be fully specified by the *Role Profile* and *Role Behaviour* elements, as specified in [1]. In order to provide a full description of a role in the form of an XML document, all actions involving role *R* are been identified. Then, for every action *a* found a new node is added to the role XML document and all protocol actions

succeeding *a* are added to it. As a result, the role schema will contain actions that the role in question is not directly involved in as we are just selecting everything succeeding action a from the protocol ontology, rather than the set of actions that the role is involved in. In our approach, only if the protocol is not enactable, additional knowledge will have to be inserted.

Blanc and Haumerlain [14] raise the issue of the agent been overloaded with big protocols if all the information is provided, and suggest the separation of knowledge in two different aspects. These would be the *strategic* aspect which is generated by the agent itself and consists of generating a strategy for the protocol (e.g. in an auction how should the agent bid) and the *participation* aspect that is about the agent actually participating in the protocol. The *participation* aspect will, effectively, realise the strategy plotted by the agent's *strategic* aspect. The protocol rules are defined as a *Petri Net* [11]. In order to retrieve the rules pertinent to the role, we replace any actions in which the role is not involved with ϵ. The idea is that every state in the Petri net will be characterised by a marking, i.e., the number of tokens on each place of the Petri net. The initial markings will make up the initial state and the transition relation is an empty set ($\emptyset$); afterwards, the *Graphe* [14] algorithm is applied. Their definition of a protocol can easily be accommodated by the games-based representation in [16]. Moreover, as we are interested in assessing the agent's competence and return to the agent the part of the protocol that it will be assuming in the society, we need the actual content of the messages rather than the Petri-net markings.

Giordano et al. [5] consider the representation of a local view (or role skeleton), as they look at the alphabet of each agent (Σ_i) separately. They are specifically interested in the actions that agent i can understand (send or receive). Any other action taken in the protocol will have a local equivalent that will be the empty action (ϵ) if the agent in question is not involved in it, either as a sender or a receiver. Also, the way that the local view of the agent is constructed is essentially by the use of $\tau^{\star}\alpha$ bisimulation, as any actions not relevant to the agent are discarded. This leads to problems, especially for protocols with a branching structure as it is not taken at all into consideration.

6 Conclusion

In this work, we looked at how a protocol specified as an LTS can be broken down into individual role automata with the use of *branching bisimulation*. However, no assumption can be made about the *enactability* of the resulting protocol. In some cases repair will be needed. We presented three approaches for repairing the protocol differentiating on the actions that need to get repaired.

This work can be expanded along with the work on the representation of the protocols in [16]. We are aiming for a representation with a higher level of abstraction, including the notion of *compound games* (i.e., describe the initial game as a composition of smaller games). If the resulting role automata can be composed in the same way that the original protocol was, it will allow for a much higher level of granularity. Furthermore, we aim to look closer into the effectiveness of the repair algorithms. We plan to perform all different repair algorithms on a number of protocols and assess the number of repairs that they will be making.

References

1. Bouaziz, W.: Une Ontologie de Protocoles pour la Coordination de Systèmes Distribués. In: Journées Francophones sur les Ontologies (JFO), Sousse, Tunisie, 18/10/07-20/10/07. pp. 231–246. Centre de Publication Universitaire (Octobre 2007)
2. Bouaziz, W., Andonoff, E.: Dynamic execution of coordination protocols in open and distributed multi-agent systems. In: Håkansson, A., Nguyen, N.T., Hartung, R.L., Howlett, R.J., Jain, L.C. (eds.) Agent and Multi-Agent Systems: Technologies and Applications, Third KES International Symposium, KES-AMSTA 2009, Uppsala, Sweden, June 3-5, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5559, pp. 609–618. Springer (2009)
3. Davidsson, P., Johansson, S.: On the potential of norm-governed behavior in different categories of artificial societies. Comput. Math. Organ. Theory 12(2-3), 169–180 (2006), http://dx.doi.org/10.1007/s10588-006-9542-x
4. Desai, N., Mallya, A.U., Chopra, A.K., Singh, M.P.: Interaction protocols as design abstractions for business processes. IEEE Transactions on Software Engineering 31(12), 1015–1027 (2005)
5. Giordano, L., Martelli, A.: Verifying agents' conformance with multiparty protocols. In: Fisher, M., Sadri, F., Thielscher, M. (eds.) CLIMA IX. Lecture Notes in Computer Science, vol. 5405, pp. 17–36. Springer (2008)
6. Goradia, V., Mowry, B., Kang, P., Panjwani, M., Lowe, D., Somogyi, A., Magruder, P., Wagner, T., McNeil, D., Yang, C., Arms, W., Sirbu, M., Tygar, D.: Netbill 1994 prototype. TR 1994-11, Information Networking Institute, Carnegie Mellon University (1994)
7. Kloukinas, C., Lekeas, G., Stathis, K.: From agent game protocols to implementable roles. In: EUMAS 08, Sixth European Workshop on Multi-Agent Systems, Bath, UK. pp. 1–15 (2008)
8. Kowalski, R., Sergot, M.: A logic-based calculus of events. New Generation Computing 4(1), 67–95 (1986)
9. McCarthy, J., Hayes, P.J.: Some philosophical problems from the standpoint of artificial intelligence pp. 26–45 (1987)
10. Milner, R.: A Calculus of Communicating Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA (1982)
11. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541–580 (April 1989)
12. Nwana, H.S.: Software agents: An overview. Knowledge Engineering Review 11(3), 205–244 (1996)
13. Sangiorgi, D.: On the origins of bisimulation and coinduction. ACM Trans. Program. Lang. Syst. 31, 15:1–15:41 (May 2009), http://doi.acm.org/10.1145/1516507.1516510
14. Sibertin-Blanc, C., Hameurlain, N.: Participation components for holding roles in multiagent systems protocols. In: Gleizes, M.P., Omicini, A., Zambonelli, F. (eds.) ESAW. Lecture Notes in Computer Science, vol. 3451, pp. 60–73. Springer (2004)
15. Stathis, K.: Game–based development of interactive systems. Ph.D. thesis, Department of Computing, Imperial College London (November 1996)
16. Stathis, K., Lekeas, G., Kloukinas, C.: Competence checking for the global E-service society using games. In: O'Hare, G.M.P., Ricci, A., O'Grady, M.J., Dikenelli, O. (eds.) ESAW. Lecture Notes in Computer Science, vol. 4457, pp. 384–400. Springer (2006)
17. Venkatraman, M., Singh, M.P.: Verifying compliance with commitment protocols. Autonomous Agents and Multi-Agent Systems 2(3), 217–236 (1999)
18. Weiser, M.: The world is not a desktop. ACM Interactions 1(1), 7–8 (November 1994), http://doi.acm.org/10.1145/174800.174801