From Agent Game Protocols to Implementable Roles

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

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

Abstract. We present a formal framework for decomposing agent interaction protocols to the roles their participants should play. The framework allows an Authority Agent that knows a protocol to compute the protocol's roles so that it can allocate them to interested parties. We show how the Authority Agent can use the role descriptions to identify problems with the protocol and repair it on the fly, to ensure that participants will be able to implement their role requirements without compromising the protocol's interactions. Our representation of agent interaction protocols is a game-based one and the decomposition of a game protocol into its constituent roles is based upon the branching bisimulation equivalence reduction of the game. The work extends our previous work on using games to admit agents in an artificial society by checking their competence according to the society rules. The applicability of the overall approach is illustrated by showing how to decompose the NetBill protocol into its roles. We also show how to automatically repair the interactions of a protocol that cannot be implemented in its original form.

1 Introduction

This paper considers a basic problem which arises when multi-agent systems (MAS) are allowed to admit new agents and, as such, need to provide these with the descriptions of the protocols within which they will be engaged. The problem is that in highly dynamic, reconfigurable and evolving systems, the protocols cannot be assumed to be correct. The protocols may have design errors in them to start with or the errors may appear due to their composition [1]. The type of errors that are usually explored are deadlocks and the possibility of a protocol to never exit a loop (liveness). However, there is another type of error which can render a deadlock-free, live protocol to be unusable in practice, because it is not implementable. The paper therefore examines this category of problematic protocols, starting with a formal framework for describing protocols and roles and then providing an algorithm to both identify non-implementable protocols and repair them at run-time, thus offering participants correct, implementable role specifications.

1.1 Motivation & Background

One of the emerging paradigms for designing and implementing complex systems is Service-Oriented Computing (SOC), which views systems as collections of services [2].

These services are specified at a rather high level, close to the business needs of the users of the services and they can be composed to form more complex services. The composition of services does not occur only at design time - instead it is required that SOC systems are fault-tolerant to service outages by being able to identify and compose with new compatible services by different providers at run-time. This run-time composition can even require the dynamic adaptation of the new services which are composed with a running system, so as to ensure that these conform to the system requirements, just as is done (at design-time) in [3, 4]. Such highly dynamic, reconfigurable systems with high-level goals are excellent candidates for being implemented as open multi-agent systems - decentralised and highly distributed systems that consist of a large number of loosely coupled autonomous agents. These agents will be pro-actively seeking ways to achieve their goals, through collaboration with agents that offer services of use to them.

Artificial Societies & Competence. As previously discussed in [5, 6], artificial agent societies are one possible way of organising and enacting the various services in a way which meets a system's requirements. These societies are composed of autonomous computational entities and their existence and function is governed by a set of protocols that determine the permissible interactions between member, as well as applicant, agents. An agent will have to become a member of a society before using the services offered in the society. Societies can be classified into *open* ones, where any agent can join and leave without restrictions of any kind, and *closed* ones, where the membership of the society is static and predefined. A third alternative, which matches better the needs of SOC systems is *semi-open* societies, where new members can join but only if they go through an application process [7], whereupon their abilities will be assessed by a society's member, called the *Authority Agent* (AA) [5]. The AA has a rather difficult problem to solve. The societal agents and protocols are most likely not designed and developed by the same person, they are implemented in different languages and based on different architectures. The AA cannot know the internal reasoning processes of the agents either, as for example in the case of BDI agents [8]. Consequently, it can rely only on its knowledge of the societal interaction protocols [9] in order to represent agent interactions and assess the ability of an applicant agent to join the society. In doing so, the AA needs to ask the applicant about its *capabilities*, i.e., what actions it is capable of doing, but not any information about when it is performing that action or under what conditions, and then decide whether the agent is *competent* to participate in the societal protocols, given these capabilities. Different definitions of competence exist [10, 11]; in our previous work [5] we have considered that competence means that it is in theory possible for the applicant to traverse all the paths of a protocol and reach all terminating states.

Protocol Representation. The representation of agent interaction protocols has been the subject of extensive research in MAS, with numerous approaches [12–14] applied to such diverse practical application areas as Semantic Web, Ambient Intelligence and Complex Systems [15]. In previous work [16], we have viewed interaction protocols as *games*, to not only acknowledge the fact that agents may be thinking strategically to achieve their goals but, also, to propose games as a metaphor for agents to communicate and coordinate their actions. This game-based representation of protocols has been used

in [10] to address the assessment of an agent's competency to join a society [6], based on the skills that the agent says it can provide. Then, in [5], we showed how to check the competency of an applicant agent, using a representation framework which allows testing the reachability of a game protocol's final states, based on action languages such as the event [17] and situation [18] calculi.

Protocol Repair & Role Derivation. However, checking whether an agent is competent is only one aspect of the problem. As aforementioned, the AA must first of all check that protocols are indeed correct, before it can check an applicant's competence for these. If the protocols are not correct then there are cases where the AA can repair them at run-time. Assuming that protocols are implementable and the applicant is competent for these, the AA will need to provide it with the description of the protocols. One possibility is for the applicant to receive the full protocol(s); that is all interactions with all other agents, no matter whether the agent is involved in these interactions. This, however, has a number of practical disadvantages:

1. the description of the whole protocol may be too large;
2. parts of the whole protocol may be irrelevant to the new agent, as they will be describing interactions of other agents;
3. for certain protocols it may be undesirable for all agents to know the whole protocol (e.g. for security reasons), so agents must work on a need-to-know basis.

As such, it is desirable that the AA can break down a protocol into its constituent roles and provide each agent with the descriptions of the roles that the agent is assuming. As such the agent uses fewer resources for representing the protocols it is engaging in, has no superfluous information of irrelevant interactions to slow down its reasoning and operates on a "need-to-know" basis.

1.2 Paper Structure

The paper is organised as follows. The next section presents a variation of the well-know NetBill [19] protocol, which will be used as a running example. Then, section 3 defines our formal framework for representing game protocols and game roles based on labelled transition systems (LTS), while section 4 presents our method for deriving independent role descriptions from a game protocol. In section 5 we discuss the case of protocols which are not implementable and present a method which can be used by the AA to repair these automatically at run-time and thus enable the society to continue using these protocols. Finally, we conclude in section 6 with a summary of our contributions and a discussion of related and future work.

2 A Variant of the NetBill Protocol

The ideas presented herein will be illustrated through a variation of the NetBill [19, 20] protocol. 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 through NetBill's gateway. These are depicted in Fig. 1 and are as follows:

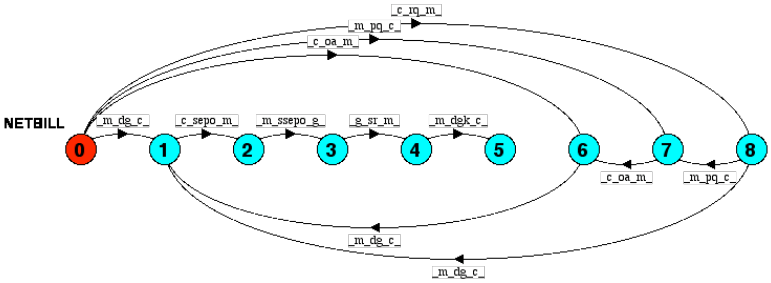

Fig. 1. A variant of the NetBill protocol

- The customer requests a quote for some digital goods from a merchant - see the transition (0, $(c, rq, \{m\})$, 8), i.e., from state 0 to state 8, labelled as $(c, rq, \{m\})$.
- The merchant provides a quote to the customer - (8, $(m, pq, \{c\})$, 7).
- The customer accepts the quote made by the merchant - (7, $(c, oa, \{m\})$, 6).
- The merchant delivers the goods encrypted with a key K - (6, $(m, dg, \{c\})$, 1).
- The customer signs an Electronic Purchase Order (EPO) with the merchant - (1, $(c, sepo, \{m\})$, 2).
- The merchant signs the signed Electronic Purchase Order and sends it to the NetBill gateway - (2, $(m, ssepo, \{g\})$, 3).
- The NetBill gateway checks the information on the EPO, transfers the money and sends the merchant a receipt - (3, $(g, sr, \{m\})$,4).
- Finally, the merchant sends the customer the key needed to decrypt the goods he purchased - (4, $(m, dgk, \{c\})$, 5).

For presentation purposes we have changed the original NetBill protocol to one with branching, assuming that there will be interactions that will require these extensions from a business point of view:

- The merchant can now make a price quote directly - (0, $(m, pq, \{c\})$, 7); e.g., for a promotional offer.
- The merchant could select to deliver the goods as its first move - (0, $(m, dg, \{c\})$, 1); e.g., when the customer has good credit.
- The customer might accept the merchant's quote directly - (0,$(c, oa, \{m\})$, 6); assuming the merchant is trusted.
- 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 - (8, $(m, dg, \{c\})$, 1); assuming a trusted customer.

3 Game Protocols as Labelled-Transition Systems

The LTS-based formal model of game protocols presented herein follows closely the situation-calculus model of [5], where a protocol describes a set of roles which interact through message-passing. The game describes the particular situations where each

message is applicable, as well as what are its effects on the state of the game. A **game protocol** ($\mathbb{P}^G$) is defined as a tuple $< P, R, S, s_0, F, \alpha, M, V, \epsilon >$, where the components are as follows. P is the protocol's name and the R component is the set of *agent roles*, which participate in the protocol. S is the set of *protocol states* and s_0 is the *initial* state of the protocol, while F is the set of *final* states of the protocol, with $s_0 \in S$ and $F \subseteq S$. Then, α are the labels of the *actions* that are known in the protocol, that is, all possible messages which can be exchanged by any two agents at any time in the protocol. The relation $M = R \times \alpha \times (2^R \setminus \emptyset)$ describes the game *moves*, i.e., associates the available actions with the role which can perform them, as well as, the roles which will be the recipients of that action, assuming that an action can be performed on multiple recipients. Finally, the relation $V = S \times M$ defines the *valid* moves for a role according to the state that the protocol is in, while the relation $\epsilon = V \times S$ defines the *effects* of the valid moves, i.e., the transition relation of the LTS.

The formal representation of the NetBill protocol of Fig. 1 will be the tuple

$$<\text{NetBill}, R, S, s_0, F, \alpha, M, V, \epsilon >,$$

where:

$$\begin{aligned}
P &= \text{NetBill} \\
R &= \{c, m, g\} \\
S &= \{0, 1, 2, 3, \cdots, 8\} \\
F &= \{5\} \\
\alpha &= \{rq, pq, oa, dg, sepo, ssepo, sr, dgk\} \\
M &= \{(c, rq, \{m\}), \cdots, (m, dgk, \{c\})\} \\
V &= \{(0, (c, rq, \{m\})), \cdots, (7, (m, dgk, \{c\}))\} \\
\epsilon &= \{(0, (c, rq, \{m\}), 8), \cdots, (4, (m, dgk, \{c\}), 5)\}
\end{aligned}$$

3.1 Modelling Roles of Game Protocols

Since the AA needs to represent the roles of a game and reason about them, we present herein a formal model of a game protocol which is built around the notion of roles and show how this is linked with the previous model of a game protocol. In a *role-oriented* model, a game protocol is assumed to be a set of roles, $\mathbb{P}^R = \{\mathbb{R}^R\}$. The descriptions of the roles are effectively projecting the components of the game protocol onto their specific role, so a role is specified as a tuple $< N, R^R, S^R, {s_0}^R, F^R, \alpha^R, M^R, V^R, \epsilon^R >$. The component R^R represents the *set of roles* that the role N will be interacting with during the course of the protocol, either as initiators of actions or as recipients of these. Then S^R are the *states* of the role, of which ${s_0}^R$ and F^R are its *initial* and *terminating* states respectively. The *actions* known to the role are α^R and the known *moves* $M^R = \{\{N\} \times \alpha^R \times (2^{R^R} \setminus \emptyset)\} \cup \{(R^R \setminus \{N\}) \times \alpha^R \times \text{has\_role}(N, 2^{R^R})\}$, where $\text{has\_role}(N, 2^{R^R})$ are all the subsets of R^R containing N, specifies whether an action can be performed by the role itself (and by which other roles it can be received)

or whether it can be performed by some other role and received by the role in question (possibly among others). The *validity* of moves according to the role's state is described through $V^R = S^R \times M^R$ and the *effects* of these moves are described through $\epsilon^R = V^R \times S^R$.

3.2 From Roles back to Game Protocols

Given the roles of a protocol, a game protocol can re easily reconstructed as follows, effectively following the *synchronous composition* of the role automata. The set of agent roles is the union of all the rolesets of the different roles. A game's state will be the combination of the states of its roles, a game's initial state will be the state where all the roles are in their initial state and a game will be terminated when all its roles have terminated. That is, any combination of terminating states of roles will be a terminating state of the game protocol. The available actions in a game will be the union of all the available actions to the roles and the same will hold for the game moves. The valid moves of the game, V, will be those where all roles involved in a move find themselves at a role state where that move is valid. That is, if a role wants to perform an action, all its recipients should be in a local state where that action is valid, i.e., we consider that same-labelled transitions in the role LTS's are synchronised. Finally, the effects of a valid move are computed by considering the effects of this move on all participating roles where it is a valid move, while the state of the other roles remains unchanged. More precisely, we have:

$$
\begin{aligned}
R &= \textstyle\bigcup_{r \in \{\mathbb{R}^R\}} {R^R}_r & S &= \textstyle\prod_{r \in \{\mathbb{R}^R\}} S_r^R \\
s_0 &= \textstyle\prod_{r \in \{\mathbb{R}^R\}} {s_0}_r & F &= \{s : \forall r \in \{\mathbb{R}^R\}.s_r \in F_r^R\} \\
\alpha &= \textstyle\bigcup_{r \in \{\mathbb{R}^R\}} \alpha_r^R & M &= \textstyle\bigcup_{r \in \{\mathbb{R}^R\}} M_r^R \\
V &= \{(s,m) : s \in S, m \in M \wedge \forall r \in \{\mathbb{R}^R\}.m \in M_r^R \Rightarrow (s_r, m) \in V_r^R\} \\
\epsilon &= \{(s_1, m, s_2) : (s_1, m) \in V \wedge \forall r \in \{\mathbb{R}^R\}. \\
&\quad \left((s_{1_r}, m) \in V_r^R \Rightarrow (s_{1_r}, m, s_{2_r}) \in \epsilon_r^R\right) \vee \left((s_{1_r}, m) \notin V_r^R \Rightarrow s_{2_r} = s_{1_r}\right)\}
\end{aligned}
$$

4 Deriving Roles of Game Protocols

As aforementioned in the introduction, the goal in deriving a role description from a game protocol is to remove all agent interactions which are of no interest to the specific role. These are the interactions that are neither initiated by the role itself nor are they received by it. Given the structure of our game protocol model, these messages can be easily identified and replaced with the invisible action τ. Actions labelled with τ in an LTS are considered to be unobservable ones and therefore other automata cannot observe them, i.e., synchronise with them. While by doing so we manage to remove unnecessary (or confidential) information, we still have not reduced the overall game protocol size. The game's states will in general be representing all the possible orderings of the different messages (including the invisible ones) which can be exchanged. In order to obtain a minimal role description we need, therefore, to minimise this automaton and obtain one which is behaviourally equivalent as far as the specific role is

concerned. This kind of required equivalence points to the use of a bisimulation equivalence reduction [21]. Such a reduction will produce an automaton which is *bisimilar* to the original one depicting the game protocol, that is, whatever traces one automaton can produce, the other can produce as well and vice-versa. The branching bisimulation equivalence (*bbe*) has been chosen among the different types of bisimulation equivalences, since it only removes invisible τ actions when doing so would not change the branching structure of the LTS. As will be shown later, this is crucial for the society's AA to be able to recognise *unimplementable* protocols, since a role's implementability depends exactly on its branching structure, i.e., the choices that the role can make. The *bbe* reduction can be easily performed on an LTS with the `ltsmin` tool of μCRL2 [22].

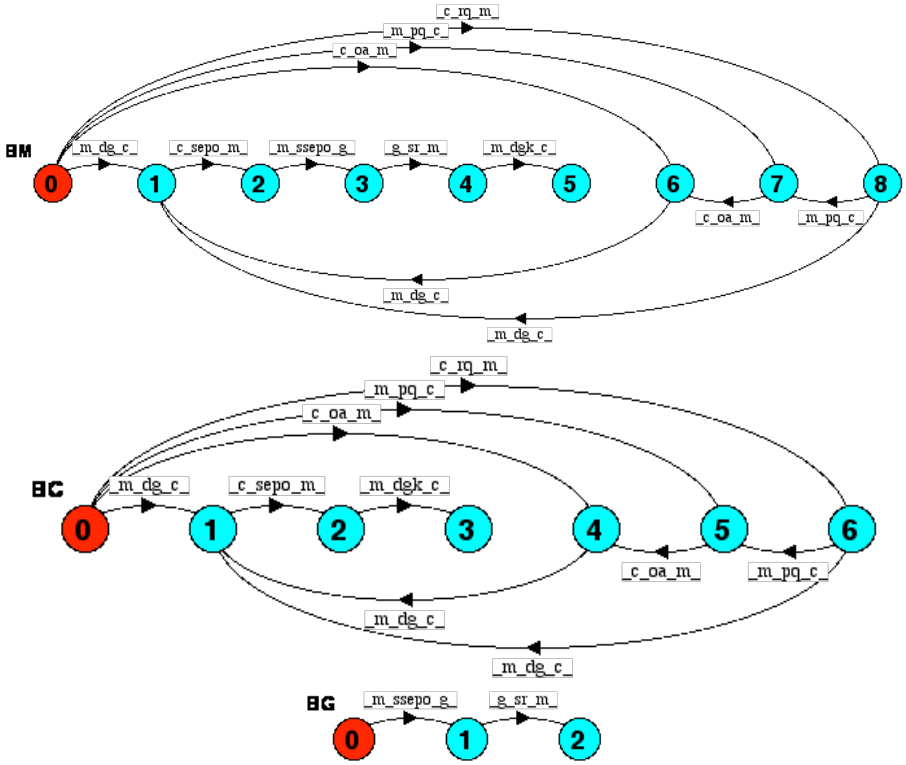

Fig. 2. Branching bisimulation results for the roles of the NetBill protocol

The results obtained by applying the *bbe* reduction to the NetBill protocol, are shown in Fig. 2. As expected, the role LTS of the merchant (BM) is not different from the original protocol, since the merchant is involved in all interactions of NetBill. However, it is immediately evident that the results for the customer (BC) and, especially, the gateway (BG) are smaller than the original protocol and do not contain any redundant information for their respective roles. When these automata are composed together they

Table 1. Deriving role attributes from a bisimulated automaton for the role

```
1  // r - role name, rg - role protocol, g - game protocol
2  compute_role_attributes(r, rg, g) {
3  roleset = states = actions = moves = valid = effects = ∅;
4    foreach (tr in rg.transitions) {
5      m = tr.move;          a = m.action;
6      sndr = m.sender;      rcvrs = m.receivers;
7      s = tr.startState; f = tr.finalState;
8      roleset = roleset ∪ {sndr} ∪ rcvrs;
9      actions = actions ∪ {a};
10     moves = moves ∪ {m};
11     valid = valid ∪ {(s, m)};
12     effects = effects ∪ {(s, m, f)};
13   }
14   foreach (s in rg.states)
15     states = states ∪ {s};
16   initial = equivalence_class(g.initialState, rg);
17   foreach (s in g.FinalStates)
18     finals += equivalence_class(s, rg);
19   return ( <r, roleset, states, initial, finals,
20             actions, moves, valid, effects> );
21 }
```

will produce the original NetBill protocol. Indeed, since the merchant's automaton is exactly the same with NetBill, the merchant will be participating in all interactions, thus constraining the overall sequence of actions.

Given the original NetBill game protocol and the automata shown in Fig. 2, we can directly obtain the role components of section 3.1, by following the algorithm shown in Table 1. The algorithm in Table 1 derives role components, such as states, roleset, moves, etc., directly from the automaton produced by the *bbe* reduction. It needs only examine the game protocol alongside the *bbe* one in order to derive the initial and final states of the role. These are the states which belong to the *bbe* class of the initial and final states of the game protocol respectively (lines 16 and 17–18). Considering the

gateway (BG), its role components are:

$$\begin{aligned}\text{Role name} &= g\\ R^R &= \{m, g\}\\ S^R &= \{0, 1, 2\}\\ {s_0}^R &= 0\\ F^R &= \{2\}\\ \alpha^R &= \{ssepo, sr\}\\ M^R &= \{(m, ssepo, \{g\}), (g, sr, \{m\})\}\\ V^R &= \{(0, (m, ssepo, \{g\})), (1, (g, sr, \{m\}))\}\\ \epsilon^R &= \{(0, (m, ssepo, \{g\}), 1), (1, (g, sr, \{m\}), 2)\}\end{aligned}$$

5 Repairing Problematic Game Protocols

The NetBill game protocol has been easily transformed into roles because it is a well designed protocol. However, it cannot be guaranteed that all protocols will be equally well designed. In fact, some may contain design errors from the beginning, while others may introduce errors as they are composed together in a society, even though they are well-designed themselves [1]. Indeed, it is a well known fact from process algebras that while safety properties are preserved under composition of processes, liveness properties are not necessarily so. So when composing two protocols which have some safety and liveness properties, their composition will have the same safety properties but it is not certain that it will have the same liveness properties - deadlocks, for example, can very easily appear due to the composition.

The errors targeted by this work, however, are beyond the classic deadlocks/liveness properties violations. Herein the interest lies in discovering the cases which make it *structurally impossible* for an agent to implement its role automaton, even though the protocol does not have a deadlock. This will occur when the agent is been asked to implement a branch in the role automaton, where at least one of the transitions is an invisible transition. What is been asked of the agent then is to take a decision without it having full knowledge of the situation. These cases are exactly the ones that the *bbe* reduction will not remove, unlike other reductions like the $\tau^* a$. The AA will be able to quickly identify these problematic protocols by examining the result of the *bbe* reduction. If there is an unobservable τ transition somewhere then the game protocol is not implementable and it should be repaired.

Fig. 3 shows one example of such an unimplementable protocol, while Fig. 4 shows the results of the *bbe* reduction for its different roles. As Fig. 4 shows, while the role Y (BY) is easily implementable, the roles Z and W (BZ and BW respectively) are not implementable at all. This is due to the fact that they both need to decide what their first action should be but that choice depends on the type of move that role X had performed before, a move that they cannot observe. As such, the AA needs to repair these problematic protocols and does so, by executing the algorithm of Table 2. Therein, it once again performs the *bbe* reduction but now it executes the algorithm of Table 3

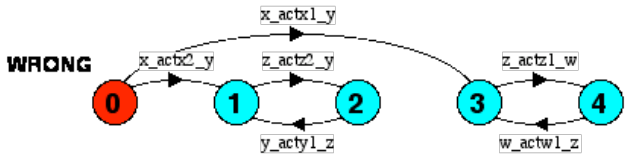

Fig. 3. A non-implementable game protocol

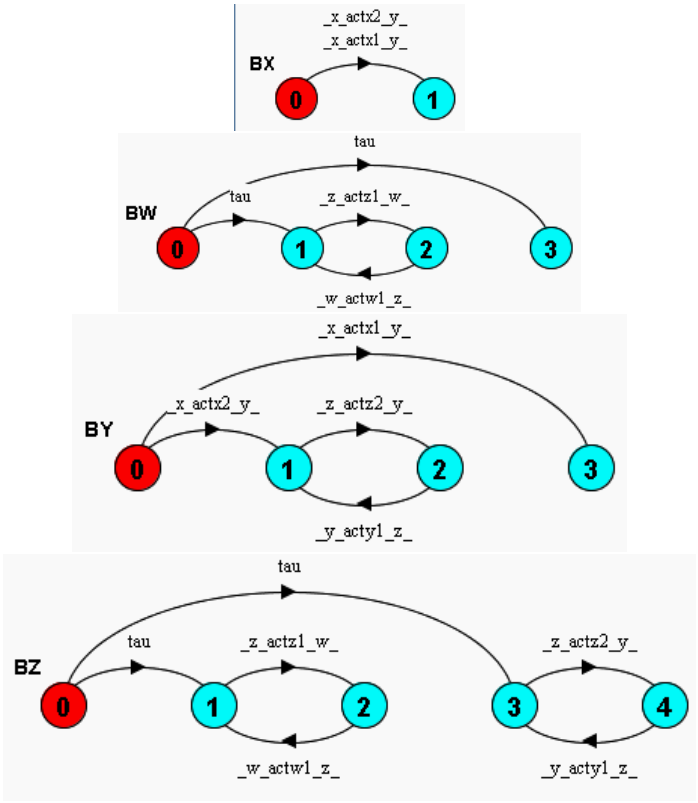

Fig. 4. The *bbe* reduced role LTS's for the unimplementable protocol of Fig. 3

to repair the game protocol when it identifies a transition in the result labelled with τ. It then iterates again, performing the *bbe* reduction on the corrected version and it only starts computing the attributes of the role when all τ transitions have been eliminated.

The repair algorithm of Table 3 starts at a state in the role protocol which has τ transitions. It uses the *bbe* classes to find out the states in the original game protocol which correspond to the starting state of these τ transitions. Then, for each of these states in the game protocol, it *adds* the current role among the receivers of the moves for all the moves which did not involve it previously. So, in the case of the game protocol of Fig. 3 and the role Z, it would replace the transition (0, (x,actx1,y), 3) with the

Table 2. Deriving role automata from a game protocol

```
1  // r - role name, g - game protocol
2  derive_role(r, g) {
3    ng = g;
4    foreach (tr in ng.transitions) {
5      if (r ∉ {tr.sender} ∪ re.receivers)
6        tr.label = tau;
7    }
8    do {
9      problematic_protocol = false;
10     ng_r = branching_bisimulation(ng);
11     foreach (tr in ng_r.transitions) {
12       if (tau == tr.label) {
13         ng = repair(ng, tr.initialState, r);
14         problematic_protocol = true;
15       }
16     }
17   } while (! problematic_protocol);
18   return compute_role_attributes(r, ng_r, g);
19 }
```

Table 3. Repairing a game protocol

```
1  //Legend: All variables refering to the Game Protocol start with GP,
2  //while all variables refering to the role protocol start with RP
3  repair(GP, RP_badState, GP_role) {
4    States []GP_class= equivalence_class(RP_badstate, GP);
5    // Add role in the receivers of the moves of these states
6    foreach (GP_state in GP_class) {
7      foreach (GP_tran from GP_state.transitions) {
8        Move GP_m= GP_tran.move;
9        Role GP_sender= GP_m.sender;
10       Roleset GP_receivers_new = GP_m.receivers ∪ GP_role;
11       GP_tran.move = Move(GP_sender, GP_m, GP_receivers_new);
12     }
13   }
14 }
```

transition (0, (x,actx1,{y,z}), 3). It would also replace the transition (0, (x,actx2,y), 1) with the transition (0, (x,actx2,{y,z}), 1). By doing so, it would give Z full information about the situation in which it is being asked to take a decision. Of course this would not completely repair the protocol, since role W would still be required to take a decision whether it should end the protocol or to wait for move (z, actz1, w) based on knowledge it does not posses. So in order to completely repair the protocol, AA would again need to add w to the receivers of the moves in the transitions (0, (x,actx1,{y,z}), 3) and (0,

(x,actx2,{y,z}), 1). The fully repaired protocol is shown in Fig. 5 and its *bbe* reduced role descriptions are shown in Fig. 6.

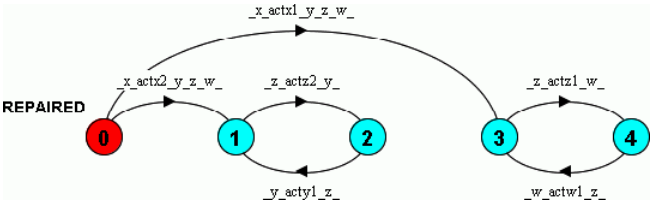

Fig. 5. The repaired protocol of Fig. 3

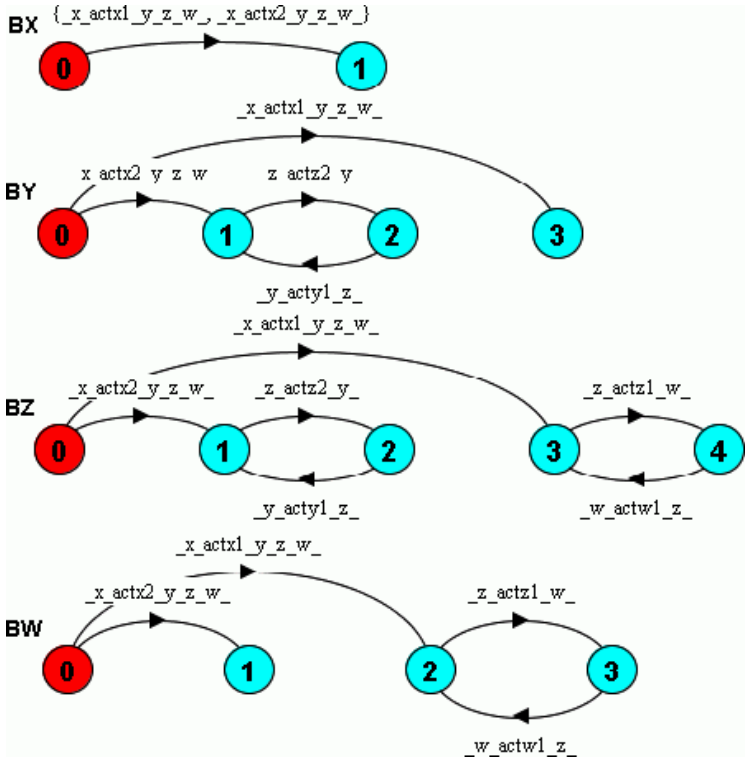

Fig. 6. The *bbe* reduced role LTS's for the repaired protocol of Fig. 5

Increasing Security. It should be noted that if the messages of the invisible actions are to be kept secret then they can be replaced with new, random strings so as not to

divulge their contents or the role which is supposed to perform them (here that is X). An extra (trusted) participant could even be added, so that the new moves are performed by it instead of the original role. So, instead of adding the move (0, (x,actx1,{y,z}), 3) we could add the moves (0, (x, act1,{y, trusted}), 5) and (5, (trusted, random1, z), 3), where trusted is the name of the new trusted participating role and random1 a random but unique action whose purpose is just to let role z know what its state should be.

6 Conclusions and Future Work

In this paper we have presented a formal framework for games which we have used to represent interaction protocols of multi-agent systems. We have shown how this framework can be viewed from a role-oriented viewpoint, by formally representing a role of a game and showing how a game description is obtained from its roles. Then we have shown how the Authority Agent of a society can use bisimulation equivalence reduction in order, first, to decide if the protocol can be broken down into its constituent roles and, then, to derive individual role descriptions for a particular game so that it can distribute these to the applicant agents. By doing so we achieve a better usage of agent resources, since they do not need to store the full game protocol, while at the same time allowing for security constraints as well, since participating agents are now working on a need-to-know basis only.

We have also shown how the Authority Agent of a semi-open society can identify problematic games, in which agents are been called to act on information they lack. In these cases we have developed a method for repairing the game with the inclusion of the minimum extra information needed, thus removing one hurdle in the interaction of agents.

Our formal framework of game protocols and role descriptions has been influenced by the situation calculus model of our previous work [5], as well as by the classic model of finite state automata ($< \Sigma, Q, q_0, F, \Delta : Q \times \Sigma \times Q >$ - actions, states, initial state, final states, transition relation) and that of an Alternating Transition System (ATS) [23], where actions are labelled by the agent which performs them ($< \Pi, \Sigma, Q, \pi : Q \rightarrow 2^{\Pi}, \delta : Q \times \Sigma \rightarrow 2^{2^Q} >$ - propositions, agents,states, labelling of states with propositions, agent transition function).

Our work on the derivation of role descriptions from game protocols has close similarities with the work of Desai et al. [24] where they also derive role skeletons from protocol specifications. However there is no consideration there of the case of problematic protocols we have identified. The authors of [24] have identified the basic problem and consider the protocols which contain non-local choices as “non-enactable”, which are exactly the protocols we call non-implementable. However, they treat these as impossible to occur in a real setting because they assume that the protocol designers will be following a set of rules to check for this kind of problem before trying to use a protocol. We believe that this is a big dependability problem for multi-agent systems where protocols of different designers are being used at the same time, since the fault of a single designer can compromise the whole system. In our approach we take what we believe to be a more realistic and cautious stance on this problem, accepting the fact that there might be some problematic protocols introduced into the system and using

a rather simple method to repair them on-line, thus allowing the multi-agent system to continue its operation unaffected by them.

Future work includes investigating the formal properties of the proposed framework to report on deriving roles and sub-roles from complex games composed of sub-games. We believe this is an interesting direction that will be found useful in many practical applications such as e-business protocols of the kind described by languages such as BPEL [25].

Acknowledgements

The authors would like to thank Dr Muck van Weerdenburg and Dr Simona Orzan for their kind help with μCRL2. All automata figures have been produced by the LTSA tool [3].

References

1. Miller, T., McBurney, P.: On illegal composition of first-class agent interaction protocols. In Dobbie, G., Mans, B., eds.: Thirty-First Australasian Computer Science Conference (ACSC 2008). Volume 74 of CRPIT., Wollongong, NSW, Australia, ACS (2008) 127–136
2. Papazoglou, M.P., Georgakopoulos, D.: Service-oriented computing. Communications of the ACM **46**(10) (October 2003) 25–28
3. Zisman, A., Spanoudakis, G.: UML-based service discovery framework. In: 4th International Conference on Service Oriented Computing, ICSOC 2006, Chicago (December 2006)
4. Kozlenkov, A., Spanoudakis, G., Zisman, A., Fasoulas, V., Sanchez, F.C.: Architecture-driven service discovery for service centric systems. International Journal of Web Services Research, special issue on Service Engineering **4**(2) (2007)
5. 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. Volume 4457 of Lecture Notes in Computer Science., Springer (2006) 384–400
6. Toni, F., Stathis, K.: Access-as-you-need: A computational logic framework for accessing resources in artificial societies. In Petta, P., Tolksdorf, R., Zambonelli, F., eds.: ESAW. Volume 2577 of Lecture Notes in Computer Science., Springer (2002) 126–140
7. Davidsson, P.: Categories of artificial societies. Lecture Notes in Computer Science **2203** (2001) 1–9
8. Morley, D.: Semantics of BDI agents and their environment. In: Proceedings of the PRICAI Workshop on Intelligent Agent Systems 1996. (1996) 119–134
9. Huhns, M.N., Singh, M.P., eds.: Readings in Agents. Morgan Kaufmann, San Francisco (1998)
10. Endriss, U., Lu, W., Maudet, N., Stathis, K.: Competent agents and customising protocols. In Omicini, A., Petta, P., Pitt, J., eds.: Engineering Societies in the Agents World IV, 4th International Workshop, ESAW 2003. Volume 3071 of Lecture Notes in Computer Science., Springer (2004) 168–181
11. Endriss, U., Lue, W., Maudet, N., Stathis, K.: Competent agents and customising protocols. In Omicini, A., Petta, P., Pitt, J., eds.: Proceedings of the 4th International Workshop Engineering Societies in the Agent World (ESAW-2003). Volume 3071 of Lecture Notes in Artificial Intelligence (LNAI)., Springer-Verlag (2004) 168–181

[3] http://www.doc.ic.ac.uk/ltsa/

12. Labrou, Y., Finin, T.W.: Semantics and conversations for an agent communication language. In: Proceedings of the Fifteen International Joint Conference on Artificial Intelligence, IJCAI 97. (1997) 584–591
13. Pitt, J., Mamdani, E.H.: A protocol-based semantics for an agent communication language. In Dean, T., ed.: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Morgan Kaufmann (1999) 486–491
14. Parsons, S., McBurney, P., Wooldridge, M.: The mechanics of some formal inter-agent dialogues. In Dignum, F., ed.: Workshop on Agent Communication Languages. Volume 2922 of Lecture Notes in Computer Science., Springer (2004) 329–348
15. Luck, M., McBurney, P., Shehory, O., Willmott, S.: Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing). AgentLink (2005)
16. Stathis, K.: A game-based architecture for developing interactive components in computational logic. Journal of Functional and Logic Programming **2000**(5) (2000)
17. Shanahan, M.: The event calculus explained. Springer Lecture Notes in Artificial Intelligence **1600** (1999) 409–430
18. McCarthy, J., Hayes, P.J.: Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., Michie, D., eds.: Machine Intelligence 4. Elsevier, New York (1969) 463–502
19. Cox, B., Tygar, J.D., Sirbu, M.: Netbill security and transaction protocol. In: Proceedings of the 1st conference on USENIX Workshop on Electronic Commerce. Volume 1., New York, New York (July 1995) 6 – 6 See also http://www.ini.cmu.edu/netbill/.
20. 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) Available from http://www.ini.cmu.edu/netbill/pubs/nb1994-11.pdf.
21. van Glabbeek, R.J., Weijland, W.P.: Branching time and abstraction in bisimulation semantics. JACM **43**(3) (May 1996) 555–600
22. Blom, S., Fokkink, W., Groote, J.F., van Langevelde, I., Lisser, B., van de Pol, J.: μCRL: A toolset for analysing algebraic specifications. In: CAV'01: Proceedings of the 13th International Conference on Computer Aided Verification. Volume 2102 of Lecture Notes in Computer Science., Paris, France, Springer-Verlag (July 2001) 250–254 See also http://homepages.cwi.nl/ mcrl/.
23. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM **49**(5) (September 2002) 672–713
24. 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) (2005) 1015–1027
25. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business process execution language for web services version 1.1. Technical report, BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems (July 2002) Last updated: February 2005. Available from http://www.ibm.com/developerworks/library/ws-bpel/.