Towards Compliance of Agents in Open Multi-Agent Systems

Jorge Gonzalez-Palacios and Michael Luck

School of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ
United Kingdom
jlgp02r@ecs.soton.ac.uk, mml@ecs.soton.ac.uk

Abstract. With the introduction of large-scale open systems, the need for managing interactions between agents, and in particular for managing the entry of a new agent into an existing system, becomes an increasingly more important objective. Without such management, there may be significant implications for the performance of such systems, negating the benefits to be gained from openness. In this paper, we sketch a process by which open multi-agent systems may be *engineered*, through the establishment of a system specification to be used by designers of agents that will enter the system, and by the system itself to check that an agent entering a system complies with the system constraints. While not fully detailed, the paper provides an initial model and a clear direction as to how such a system may be constructed, offering a new way of developing open multi-agent systems.

1 Introduction

The number of computers and computational devices has increased significantly in the last few years and, since these devices rarely work on their own, the number of networks has also exploded. In software, new technologies such as pervasive computing and the Grid are also emerging and take advantage of these networks. These technologies have brought challenging problems in computer science and software engineering, since they demand systems that are highly distributed, proactive, situated and *open*.

An open system is one that allows the incorporation of components at runtime that may not be known at design time. Usually, the components of an open system are not designed and developed by the same group, nor do they represent the same stakeholders. In addition, different groups may use different development tools and may follow different policies or objectives, thus leading to *heterogeneous* systems. Regardless of how and by whom a component is developed, it typically has the same rights to access the facilities provided by the system, as well as the obligation to adhere to its rules.

The introduction of large-scale open systems of this kind is likely to lead to a new set of problems, however, relating to the effects of interactions between

agents. Indeed, what we are beginning to witness is the emergence of computational societies, of electronic organisations, and of all the variety of good and bad consequences that they bring with them. Just as in human societies, we need to consider the impact of regulations and their absence, of opportunistic and malicious behaviour, and we need to find ways to organise and manage systems in order to mitigate their potential deleterious effect on a system as a whole. While some work has been done on each of these concerns, their combination in large-scale open systems has not been addressed, yet they are fundamental requirements if the visions of Grid computing, for example, are to be realised.

However, traditional approaches (e.g., object-oriented and component-based computing) have fallen short in *engineering* this type of application because they operate at too low a level of abstraction. For example, object-oriented computing decomposes a system into entities (or *objects*) that encapsulate information and functionality. This information, however, usually refers to basic data structures or to other objects. Similarly, the functionality of objects relies on simple procedures like those normally found in most programming languages. Elaborated object decompositions, although possible, tend to make it difficult to understand and design applications that involve high-level concepts such as grid services and workflows.

In response, different approaches have been attempted to facilitate the development of such complex applications. In particular, some evidence suggests that the multi-agent approach provides adequate abstractions to successfully develop this type of system [6], and this has resulted in the appearance of several agent-oriented software methodologies which claim to support the construction of open systems. Although agent-oriented software methodologies exist to support the development of open systems, they are lacking when dealing with the incorporation of new components (or agents) to an existing system. In particular, these methodologies do not address two different but very related problems:

- how to specify the facilities provided by the existing system for those interested in the development of new agents; and
- how to design and construct mechanisms to ensure that the integrity of the system is not violated at run-time by new agents.

Solving these problems requires the accomplishment of some non-trivial tasks. In order to solve the first problem of specifying the facilities provided by the system, we must first accomplish the selection of appropriate abstractions on which to base the specification. For the second problem of ensuring that the integrity of the system is not violated at run-time, mechanisms for monitoring the behaviour of the system and evaluation of its characteristics must be provided.

Although complete solutions to these problems are highly application and platform dependent, we can, nevertheless, separate more general problems from more specific ones and provide partial solutions. In particular, in order to create agents that are eventually incorporated into an existing system, developers need to know what facilities are provided by the system, and the way in which they can access them, so that they can design new agents in accordance with these characteristics. In addition, developers must be aware of the rules of behaviour

of the system, and design new agents in such a way that those rules are observed at run-time. From the perspective of maintaining the integrity of the system, this is particularly important in the case of multi-agent systems, because the autonomy and pro-activity exhibited by agents can easily lead to unexpected behaviour.

In this paper we present an initial model for the specification of open multi-agent systems based on organisational concepts, and then take some first steps in applying it to create a mechanism for checking that a specification is observed at run-time. In Section 2, we analyse the characteristics of a specification in open multi-agent systems, that is, *what* must be included, and *how* to express it. Then, in Section 3 we formalise such a specification. The next sections address the problem of how to check that such a specification is observed at run-time, focussing in particular on checking that the protocol used complies with the system specification. Finally, we present some conclusions.

2 Specification of open multi-agent systems

We now move to a consideration of creating specifications of open systems, in such a way that potential participants can determine the requirements and benefits of joining the system. In our case, the targets of such a description are the designers of the agents, so we do not address the problem of *adaptive* agents, although some of the considerations presented here may also apply in the case that the new agents automatically adapt to the specification at run-time.

Such a specification must be as neutral as possible, since the agents might be developed with varied techniques. However, at least some basic assumptions must be made; in particular, the use of some common, appropriate concepts is required. We use *role*, *protocol* and *organisation* as the basic concepts on which a specification can be constructed. In general, these abstractions appropriately model the characteristics found in multi-agent applications, and they give rise to a set of models that provide the documentation necessary both for developers and for automatic compliance monitoring in order for agents to join open systems in effective and managed ways. (Other approaches for controlling the behaviour of open multi-agent systems are based on fewer abstractions, for example agent interaction or protocols [8, 7]. Although this might result in less restrictions about how agents are implemented, it tends to complicate the specification of certain type of restrictions, such as those referring to the number of times a role is permitted to be enacted.)

2.1 Participants model

The participants model contains the description of each agent of the system, referring only to those individual characteristics that do not involve interaction with other agents, and that are independent of how the agent is implemented. Since we model agents by means of roles then, according to the characterisation

of roles we employ, the participant model consists of the set of roles in the system and, for each of them, a list of their services and non-functional requirements.

Services are tasks that a role can perform without interacting with other roles. We propose a simple characterisation of a service consisting of a name, the role to which the service belongs, its input and output parameters, and a description of the task itself. Since the actual implementation of the process is not restricted by the specification, its description can be text, pseudocode or any formal description. Regarding the non-functional requirements, we follow a simple approach consisting of representing each requirement by an identifier-value pair, for example $(memory, 40)$, where the identifiers and their possible values have previously been defined.

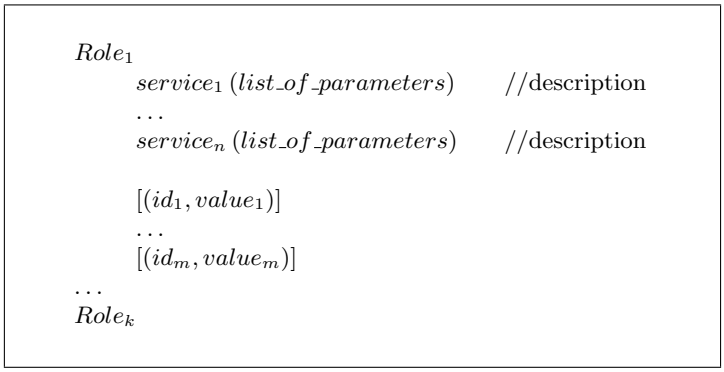

Fig. 1. The general form of the participants model

The general form of the participants model is shown in Figure 1, in which requirements identifiers are denoted by id_i and their corresponding value by $value_i$. The square brackets indicate that the use of non-functional requirements is optional. As an example, Figure 2 presents a fragment of the participants model corresponding to a Conference Management System (for which no explanation is needed), but lack of space prevents us from presenting the complete example. This simple example shows three participants, each one having a service. (Note that in this section we use different fonts in figures, to differentiate the general form of a model from the corresponding example.)

2.2 Interactions model

The interactions model describes the way roles interact by means of protocols. Our protocol characterisation is inspired by a simplified version of *sequence diagrams* similar to those of AUML, and represents the participating roles in the protocol, the messages they exchange, and the sequence of those messages. The messages are labelled with their communicative act and content, or with an identifier (whose communicative act and content are defined elsewhere, e.g. in [4]).

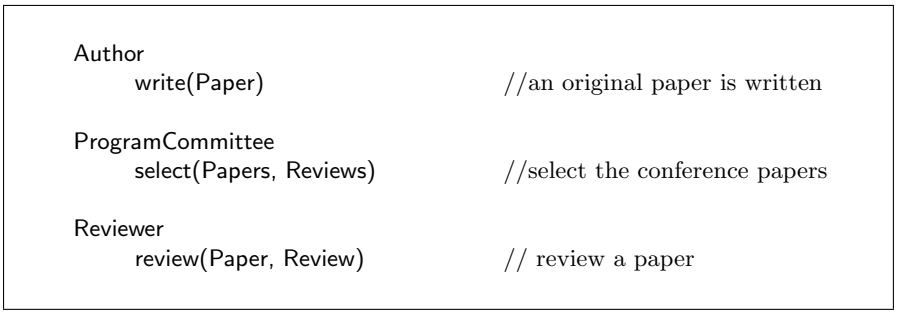

Fig. 2. The application of the Participants Model to the CMS example

The communicative acts must be described in the agent communication language specified in the Agent communication language layer. In the same way, the content must belong to the content language specified in the Content language layer and the specification of general concepts.

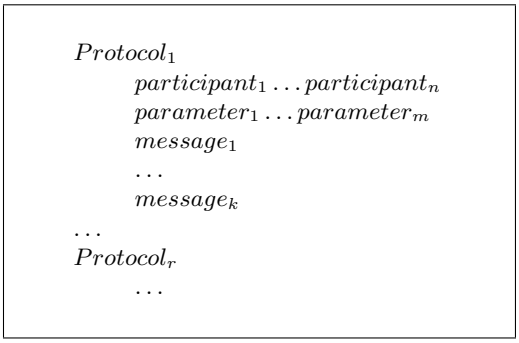

Fig. 3. The general form of the interactions model

Figure 3 shows the general form of the interactions model, in which each of the messages in the protocol is formed of a sender, a receiver, a communicative act and a content. An example showing a fragment of the interactions model for the Conference Management System is presented in Figure 4, which contains two protocols, *SubmitPaper* and *ReviewPaper*. For each protocol, the first line contains the list of participants, the second line its parameters, and from the third line on, the messages. The *ReviewPaper* protocol, for instance, involves roles *ProgramCommittee* and *Reviewer*, has parameters *paper* and *review*, and employs two messages.

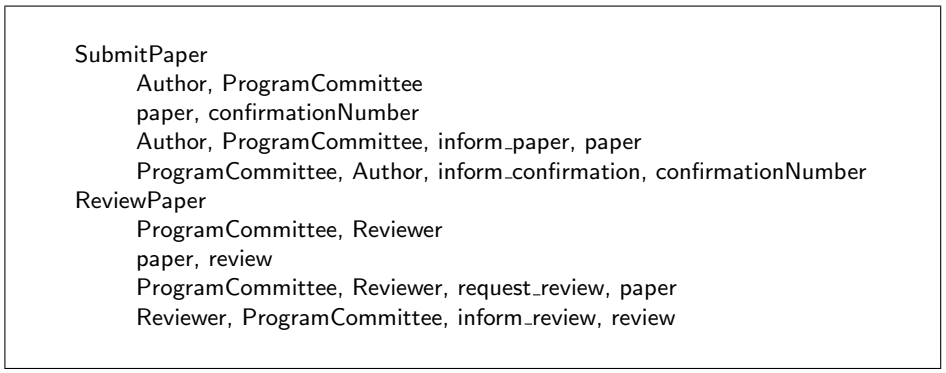

Fig. 4. The application of the Interactions Model to the CMS example

2.3 Social constraints model

The specification of social constraints contains the restrictions imposed on the agents' social behaviour. Such restrictions are represented by means of organisational concepts, more specifically by organisational rules. Organisational rules are key to the definition of the organisation and thus of the system itself. For this reason, an agent attempting to join an existing system must be provided with the set of rules it must adhere to. The specification of social constraints is formed from the list of organisational rules of the system, expressed in some appropriate language (which we will not consider in this paper because of space constraints). The general form of this model is represented in Figure 5, and an example consisting of two rules is shown in Figure 6. In the latter figure, the first rule states that there must be at least five reviewers, while the second rule states that the program committee must not assign a paper for review, to the same reviewer, more than once.

Fig. 5. The general form of the social constraints model

2.4 Summary

Up to this point, in this paper, we have focused on the creation of a system specification. Based on the results obtained here, in the following sections we

```
card(Reviewer) >= 5

For all p:paper, r: Reviewer, w: review
      card(ReviewPaper(ProgramComittee, r, p, w)) <= 1
```

Fig. 6. The application of the Social Constraints Model to the CMS example

explore the problem of ensuring that what is stated in the specification is observed at run time. Roughly, our approach consists of checking that the actions performed by an agent do not violate any of conditions stated in the sections of the specification. However, before proceeding, we formalise a specification and consider the problem of examining that the specification is complete and free of inconsistencies.

3 A model of open systems

In an open multi-agent system specification the details of the internal structure of the agents are not important, but only their functionality. This is because the agents in the system may be constructed by different developers and following different techniques. For the same reason, the implementation details of the protocols are not relevant, but only their patterns of interaction. This ensures that the agents can be developed according to different tools if they comply with the rules of the system. In this section we present a formal model for open multi-agent systems, based on organisational concepts, and that abstracts the functionality of the agents and the way they interact, regardless of implementation issues.

We define a model for an open multi-agent system as a tuple $\langle \mathcal{E}, \mathcal{N}, \mathcal{P}, \mathcal{S}, \mathcal{O} \rangle$, where:

1. $\mathcal{E}$ is a 4-tuple of set of elements of the system;
2. $\mathcal{N}$ is the set of the *roles' non-functional requirements*;
3. $\mathcal{P}$ is the set of *protocols*;
4. $\mathcal{S}$ is the set of *services*; and
5. $\mathcal{O}$ is the set of *social constraints*.

3.1 Elements of an open system

$\mathcal{E}$, the tuple of elements in the system, has the form $\langle R, P, S, D \rangle$, where each entry is a set whose elements are identifiers, as follows:

1. R is the set of role identifiers;
2. P is the set of protocol identifiers;
3. S is the set of service identifiers; and
4. D is the set of identifiers of general concepts, which are resources and entities of the environment that are used in the description of other parts of the model, such as protocols, services and social constraints.

3.2 Non-functional requirements

The elements of the set $\mathcal{N}$ have the form (r, n, v), where $r \in R$, n denotes a type of non-functional requirement, and v represents a possible *value* of n. The interpretation of this is that such a role requires at least that value for the non-functional requirement in order to be played. For example, in the conference management system,

$$(\mathit{ProgrammeCommitteeChair}, \mathit{confidentiality}, \mathit{1})$$

indicates that the role *Chair* must comply with the highest (1) confidentiality. However, it must be noted that the list of non-functional requirements and their associated values are highly dependent on the application and platform used.

3.3 Protocols

Each element of $\mathcal{P}$, the set of protocols, is a 5-tuple of the form (p, I, C, A, M), where:

1. $p \in P$ is a unique protocol name,
2. $I \in R$ is the initiator of the protocol,
3. $C \subset R$ is the set of collaborators, that is, the roles that participate in the protocol, apart from the initiator,
4. $A \subset D$ is the set of input and output parameters,
5. M is the allowed sequence of messages, expressing the order the messages must follow during the execution of the protocol. This is a sequence of instructions, each of which is either a message or a *compound message*. A compound message encompasses a *connector* and a *set* of messages, and represents the concurrency connectors of AUML. Concurrency connectors are used as a means to express that multiple messages are sent at the same time, and are of three types: *and* (AND), *inclusive or* (OR), and *exclusive or* (XOR). In the first case all the messages are sent in parallel, while in the second zero or more messages are sent and in the last case only one message is sent.
   Finally, each element of M, the set of messages of a protocol, has the form (r_s, r_r, b), where:
   $r_s \in R$ is the sender;
   $r_r \in R$ is the receiver; and
   b is the body of the message.

3.4 Services

$\mathcal{S}$, the set of services, consists of elements of the form (s, r, B), where:
$s \in S$ is a unique service name, and
$r \in R$ is the role to which the service belongs,
$B \subset D$ is the list of parameters of the service.

3.5 Social Constraints

$\mathcal{O} \subset \mathcal{L}$, the set of social constraints, contains the expressions that govern the function of the system.

Table 1 summarises this notation. For simplicity, we do not include the part corresponding to the sequence of messages, but only the structure of each message.

$\mathcal{E}$	element identifiers		
	R	role identifiers	
	P	protocol identifiers	
	S	service identifiers	
	D	concept identifiers	
$\mathcal{N}$	non-functional reqs.		
	r	role to which applies	
	n	non-functional reqs. identifier	
	v	value	
$\mathcal{P}$	protocols		
	p	protocol identifier	
	I	initiator	
	C	collaborators	
	A	protocol parameters	
	M	sequence of messages	
		For each message:	
		s_e	sender
		s_r	receiver
		b	body
$\mathcal{S}$	services		
	s	service identifier	
	r	role	
	B	service parameters	
$\mathcal{O}$	social constraints		

Table 1. Summary of notation

4 Compliance monitoring

A specification describes a system from different perspectives; for example the specification of protocols deals with the interaction aspects while the specification of participants focuses on the individual aspect of roles. However, it is essential that these perspectives are not in contradiction, but describe the system in a *consistent* form. For instance, an organisational rule cannot reference a protocol that has not been defined in the specification of interaction protocols. For this reason, we need a mechanism for checking consistency in the specification. Such a mechanism can be implemented in different ways; for example, by

means of a software tool the consistency can be checked every time the specification is updated. Whatever the mechanism used, the following conditions must be checked.

1. The name of roles, protocols, responsibilities and general concepts must be unique.
2. All the protocols mentioned in the specification must be described in the specification of interaction protocols.
3. All the roles mentioned in the specification participate in at least one protocol and have at least one responsibility.
4. All the resources mentioned in the specification must be defined in the specification of general concepts.

5 Static analysis on agent entry

As mentioned above, our approach to the problem of ensuring the integrity of an open system is to check, at run-time, that what is stated in the specification is not violated. In other words, we are assuming that the integrity of a system is ensured if all the conditions expressed in the specification are observed. On the other hand, since organisational rules form a part of a specification, and the purpose of organisational rules is to ensure the correct behaviour of the system, it follows that the observance of the specification can also ensure that the system behaves as expected at run-time. This is particularly helpful when no other methods of verification or validation are used.

Another assumption that we make deals with how agents enter a system. We assume that each time an agent attempts to enter the system, a mechanism is used to decide whether its entry is accepted. Also, if they are accepted, and during their lifetime, agents can play roles, or quit playing roles. Both actions are notified to the system, and the former needs authorisation too.

Based on the assumption above, and with the aim of monitoring their behaviour, we divide the functionality of an agent into *static* and *dynamic*. Static functionality occurs during the entry of the agent or when a role is assigned to the agent, whereas dynamic functionality occurs (perhaps additionally) at any other moment, for example a protocol initiation.

To illustrate this point, suppose that an agent intends to enter the system. It must first receive approval from a run-time component, hereafter called the *monitor* and depicted in Figure 7. As suggested in the figure, the only way for an agent to access the system is by getting approval from the monitor, based on the *characteristics* of the agent, and the specification of the system. By developing such a monitor, we provide a means to verify statical functionality, for example to detect if an agent's protocol has an incorrect *initiator role*, in the sense that it does not match what is stated in the specification. However, the monitor does not consider aspects that are not verifiable statically, such as if a protocol is executed at the wrong moment.

With these considerations in mind, we proceed to analyse how to check the observance of a specification.

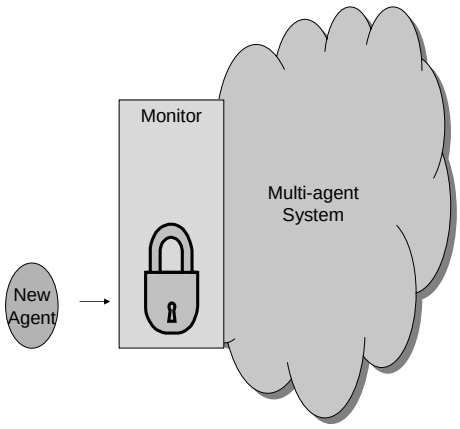

Fig. 7. The function of the monitor

5.1 Run-time participants analysis

The run-time analysis for the participants has the aim of ensuring that the agents comply with the participants model of the specification. This can be done statically, at the moment the agent requests authorisation to play a role. Note that the agent can be playing other roles, or no role at all before attempting to play a specific role. When an agent requests authorisation to play a role, the monitor must check that the characteristics of the agent, and the way it implements the role, match the conditions stated in the participants model. More specifically, given the role in question, the services as implemented by the agent, and the resources that the agent possesses, the monitor must check that the following conditions hold.

- The role that the agent intends to play exists and is available; that is, the role has not exceeded its cardinality.
- The agent has enough resources to satisfy each of the non-functional requirements specified in the participants model.
- The agent implements all the services specified in the model, and in the way they are specified, in terms of name and parameters.
- Optionally, for a stricter checking, the agent does not implement other services apart from those specified in the model.

Note, however, that checking the services in this way only offers a guarantee that their interfaces have been correctly implemented, but does not say anything about whether their implementation is *semantically* correct; for example, if instead of adding two numbers, they are multiplied.

5.2 Run-time protocol analysis

During the entry of an agent to the system, we can also check, to some extent, whether the protocols implemented by the agent correspond to those specified.

Essentially, the procedure is a matter of matching the characteristics of both protocols: those of the agent implementation and those specified in the system. Most of the checking is straightforward, except the part regarding the sequence of messages of the protocol, which depends on how many features of the sequence diagrams are considered. According to this, the algorithm is divided into two parts: matching the head and matching the messages. Protocols are accepted only if they are accepted in both parts. However, it must be noted that this procedure does not check the dynamic characteristics of the protocol, such as the *actual* sequence in which the messages are sent, nor the actual content of the messages, since there is no mechanism to guarantee that the characteristics of the protocol, as were checked, are observed during the operation.

In the following, such a procedure, together with its inputs and outputs, is presented.

Algorithm for matching the head The matching the head part deals with checking that the role exists and that the protocols correspond to those specified in the interactions model. The interactions model was presented in Section 2.2, and is refined below using a notation that is more appropriate for expressing the algorithm.

Let $R = \{r_1, r_2, \ldots, r_k\}$ be the set of roles of the system (where k is the number of roles), and
Q_i the set of protocols associated to role r_i.
Since Q_i contains the protocols associated with role r_i, it can be expressed as $Q_i = \left\{q_1^i, q_2^i, \ldots, q_{m_i}^i\right\}$, where m_i is the number of protocols associated with role i, and each q_j^i denotes a protocol and thus have the form
$q_j^i = \left(p_j^i, I_j^i, C_j^i, A_j^i, M_j^i\right)$, where:
p_j^i is the name of the protocol,
$I_j^i \in R$ denotes the initiator,
$C_j^i \subset R$ denotes the collaborators,
A_j^i is the (ordered) sequence of parameters of the protocol, each consisting of a *name* and a *type*, so we can express it as
$A_j^i = \left\langle (a_1, t_1), (a_2, t_2), \ldots, \left(a_{m_j^i}, t_{m_j^i}\right)\right\rangle$, where m_j^i is the number of parameters of the protocol, and finally
M_j^i is the sequence of messages.

The algorithm is presented in Figure 8 and, as can be observed, is straightforward and consists of checking the compliance of the protocol name, the initiator, the collaborators and the sequence of parameters of the protocol.

Algorithm for matching the messages In the second part of the procedure, matching the messages, the objective is to check that the sequence of messages stated in the specification is equivalent to the sequence of messages implemented by the agent, so that any possible difference in the expression of the protocol is not important for the execution. (From this perspective, we can ignore several

Inputs:

r the role in question; and
$Q \subseteq \mathcal{P}$, the set of protocols involving r, as implemented by the agent

Output:

acceptance:

true if the header of the protocol complies with the specification;
false otherwise

Algorithm:

acceptance = $false$
$r \notin R \Rightarrow$ **exit**
$\exists e$ such that $r = r_e \wedge 1 \leq e \leq m$
$\forall\, (p, I, C, M) \in P_r$
$p \notin \{p_1^e, p_2^e, \ldots, p_{m_e}^e\} \Rightarrow$ **exit**
$\exists t$ such that $p = p_t^e \wedge 1 \leq t \leq m_e$
$I \neq I_t^e \Rightarrow$ **exit**
$C \neq C_t^e \Rightarrow$ **exit**
$\forall\, (a, y) \in M$
$(a', y') = nextElement\,[M_t^e]$
$a' \neq a \vee y' \neq y \Rightarrow$ **exit**
acceptance = $true$

Fig. 8. Algorithm: Matching the Head

features of sequence diagrams, but we do have to consider some others which are relevant when describing a sequence of messages.)

Before proceeding with the algorithm, it is worth mentioning the extent of the algorithm in terms of how the sequence of messages is formed. Our representation of protocols is based on AUML sequence diagrams, which are rich in features, some inherited from UML sequence diagrams and some exclusive to agents. Specifically, we must check the multiplicity of the messages — that the number of messages sent and the number of receivers of the messages must correspond to those of the specification — and the type of message delivery — that whether it is synchronous or asynchronous, it must match that specified in the system. We consider two types of structures: conditions and concurrency connectors. A condition is a logical expression that determines if a message is sent or not. As was mentioned before, concurrency connectors are used as a means to express that multiple messages are sent at the same time and are of three types: *and* (AND), *inclusive or* (OR), and *exclusive or* (XOR).

However, for our purpose (checking whether two sequence diagrams represent essentially the same protocol) not all the features are relevant. While we need to consider the roles involved in the protocol and their existence in the system, and the *and*, *or* and *exclusive or* parallel connectors, the conditions of messages

can be ignored since they are meaningful only at execution time. In particular, we do not consider: agents, since we only allow roles as participants of protocols; lifelines and threads of interaction, since they are not relevant in the functionality of the protocol; nested and interleaved protocols, since they are not considered in our definition of protocol; and protocol templates, for the same reason.

Since this algorithm is meant to be executed statically, it simply checks that the sequence of messages matches the sequence specified in the system, but in the case of messages joined by a concurrency connector, the messages can appear in any order. Conditions are just ignored as they are relevant only at run-time.

To describe this algorithm we make use of the following functions. The first two functions operate on a message instruction, while the last two operate on a compound message. The *message* function returns *true* if and only if the message instruction is a simple message, and not joined to other messages by a concurrency connector. The *compound_message* function returns *true* if and only if the message instruction is a compound message, (a set of messages joined by a concurrency connector). The *connector_of* function denotes the concurrent operator of a compound message ($\in \{AND, OR, XOR\}$). Finally, the *set_of_messages* function denotes the set of messages of a compound message. Note that this function denotes a set, not a sequence, since the order of the messages is not important.

The algorithm is presented in Figure 9. As can be observed, for the protocol to be accepted, the messages are compared. Simple messages are examined for equality, whereas for compound messages of type OR and XOR, equality is not required, but being a subset is enough. We have implemented this algorithm by translating the sequences of messages into non-deterministic finite state machines and then checking their equivalence.

6 Related Work and Conclusions

It has been argued by many [2, 3] that agents interacting in a common society need to be constrained in order to avoid and solve conflicts, make agreements, reduce complexity, and in general to achieve a desirable social order. This is the role of norms, or organisational rules, which represent what ought to be done by a set of agents, and whose fulfilment can be generally seen as a public good when their benefits can be enjoyed by the overall society, organisation or group [1]. Research on norms and agents has ranged from fundamental work on the importance of norms in agent behaviour—[3], to proposing internal representations of norms [10], considering their emergence in groups of agents [13], and proposing logics for their formalisation [9]. Despite such efforts to understand how and why norms can be incorporated into agents and multi-agent systems, there is still much work to do, particularly in relation to the engineering of such norm-based or organisation-based systems.

The easiest way to represent and reason about norms is by seeing them as built-in constraints where all the restrictions and obligations of agents are obeyed absolutely without deliberation. In this view, the effort is left to the system

Inputs:

$S = \langle m_1, m_2, \ldots, m_n \rangle$, the sequence of specified messages
$S' = \langle m'_1, m'_2, \ldots, m'_n \rangle$, the sequence of implemented messages

Output:

acceptance

Algorithm:

acceptance = **false**
$\forall i \in \{1, \ldots, n\}$
  $message\,(m_i) \Rightarrow$
    $m_i \neq m'_i \Rightarrow$ **exit**
  $compound\_message\,(m_i) \Rightarrow$
    $connector\_of\,(m_i) =$ **AND** $\wedge$
      $set\_of\_messages\,(m'_i) \neq set\_of\_messages\,(m_i) \Rightarrow$ **exit**
    $connector\_of\,(m_i) \in \{$**OR**, **XOR**$\} \wedge$
      $\neg\,(set\_of\_messages\,(m'_i) \subseteq set\_of\_messages\,(m_i)) \Rightarrow$ **exit**
acceptance = **true**

Fig. 9. Algorithm: Matching the Messages

designer to ensure that all agents respond in the required way and, consequently, that the overall system behaves coherently. However, this may result in inflexible systems that must be changed off-line when either the agents or the environment change. By contrast, if a dynamic view of norms is taken, the flexibility of the overall system can be assured [14]. We have considered the role of norms in the design of agent architectures and their reasoning processes elsewhere, but in this paper we have focussed on the *engineering* of such systems through the use of organisational rules, specification and compliance. It is precisely the use of organisational rules that distinguishes our specification from other organisational models such as [11, 12, 5].

In contrast to other models for the development of multi-agent systems based on organisational concepts, the model presented in this paper excludes any reference to particular implementation issues, and so is suitable for *open* multi-agent systems. In addition, this model uses organisational rules, which makes it capable of handling situations that other approaches leave unconsidered, such as restrictions about the number of times a role must be enacted. Finally, our model also describes the facilities provided by a system, and they way they are accessed, which is essential for the development of new components, and their eventual incorporation into the system.

In other work, in addition to considering compliance with protocols, we also consider other elements of the specification that can be checked at the entrance of an agent to the system, which are the participants model, and a specific subset of organisational rules. The former includes checking the services and non-

functional requirements of the roles in the system. The latter refers to checking the compliance of those rules that refer only to static properties of role assignment such as the number of times a roles must be played, sequence in which roles must be played, and conflicts in playing more that one role at the same time.

Also, in other work, we consider the compliance of a more general type of organisational rules, this is, those rules that need to be checked continually, not only at the entrance of an agent to the system. For this, we have designed a mechanism that collects the relevant information from the system and analyses the corresponding rules. This work presents some similarities to works on law enforcement [8, 7], being the main difference (in addition that we use *organisational rules* and they use *laws*) that our work consists in monitoring compliance, while theirs consists in preventing violation.

References

1. C. Castelfranchi, R. Conte, and M. Paolucci. Normative reputation and the cost of compliance. *Journal of Artificial Societies and Social Simulation*, 1(3), 1998.
2. R. Conte. Emergent (info)institutions. *Journal of Cognitive Systems Research*, 2:97–110, 2001.
3. R. Conte, R. Falcone, and G. Sartor. Agents and norms: How to fill the gap? *Artificial Intelligence and Law*, 7(1):1–15, 1999.
4. FIPA. http://www.fipa.org/, 1999.
5. D. Grossi, F. Dignum, V. Dignum, M. Dastani, and L. Royakkers. Structural aspects of the evaluation of agent organizations. In *Proceedings of the Workshop on Coordination, Organization, Institutions and Norms in Agent Systems*, 2006.
6. Nicholas R. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001.
7. Naftaly Minsky. Law governed interaction (lgi): A distributed coordination and control mechanism. Technical report, Rutgers University, 2005.
8. R. Paes, G. Carvalho, C. Lucena, P. Alencar, H. Almeida, and V. Silva. Specifying laws in open multi-agent systems. In *Agents, Norms and Institutions for Regulated Multi-agent Systems (ANIREM)*, 2005.
9. A. Ross. *Directives and Norms*. Routledge and Kegan Paul Ltd, 1968.
10. R. Tuomela and M. Bonnevier-Tuomela. Social norms and agreements. *European Journal of Law, Philosophy and Computer Science*, 5:41–46, 1995.
11. Wamberto Vasconcelos, Mairi McCallum, and Tim Norman. Modelling organisational change using agents. Technical Report AUCS/TR0605, Department of Computing Science, University of Aberdeen, 2006.
12. Luis Erasmo Montealegre Vzquez and Fabiola Lpez y Lpez. An agent-based model for hierachical organizations. In *Proceedings of the Workshop on Coordination, Organization, Institutions and Norms in Agent Systems*, 2006.
13. A. Walker and M. Wooldridge. Understanding the emergence of conventions in multi-agent systems. In V. Lesser and L. Gasser, editors, *Proceedings of the International Conference on Multi-Agent Systems*, pages 384–389, 1995.
14. F. Zambonelli, N. Jennings, and M. Wooldridge. Organisational abstractions for the analysis and design of multi-agent systems. In *Proceedings of the First International Workshop on Agent-oriented Software Engineering*, 2000.