A Top Down Approach for MAS Protocol Descriptions*

Joaquín Peña
Dept. Lenguajes y Sistemas Informáticos
University of Seville
Avda. Reina Mercedes s/n
41012 Seville Spain
joaquinp@lsi.us.es

Rafael Corchuelo
Dept. Lenguajes y Sistemas Informáticos
University of Seville
Avda. Reina Mercedes s/n
41012 Seville Spain
corchu@lsi.us.es

José L. Arjona
Dept. Lenguajes y Sistemas Informáticos
University of Seville
Avda. Reina Mercedes s/n
41012 Seville Spain
arjona@lsi.us.es

ABSTRACT

When the protocol of a complex Multi-Agent System (MAS) needs to be developed, the *top–down approach* emphasises to start with abstract descriptions that should be refined incrementally until we achieve the detail level necessary to implement it. Unfortunately, there exist a semantic gap in protocol description methodologies because most of them first identify which tasks have to be performed, and then use low level descriptions such as sequences of messages to detail them. In this paper, we propose an approach to bridge this gap. We model MAS protocols using several abstract views of the tasks to be performed, and provide a systematic method to simplify them. Tasks are represented by means of interactions that may be refined into lower–level interactions with the techniques proposed in this paper (simpler interactions are easier to describe and implement using message passing.) Unfortunately, deadlocks may appear due to protocol design mistakes or due to the refinement process. Thus, we also propose an algorithm to ensure that protocols are deadlock free.

Keywords

Top-down approach, agent protocol descriptions, interaction refinements, and deadlock detection.

1. INTRODUCTION

Agent-Oriented Software Engineering (AOSE) is paving the way for a new paradigm in the Software Engineering field. This is the reason why a large amount of research papers on this topic are appearing in the literature. One of the main research lines in AOSE arena is devoted to developing methodologies for describing interaction protocols (hereafter protocols) between agents.

*The work reported in this article was supported by the Spanish Interministerial Commission on Science and Technology under grant TIC2000-1106-C02-01.

1.1 Motivation

When the protocol of a large MAS has to be developed, it is desirable to start with an abstract description that can be refined incrementally according to the *top down approach*[15]. In our opinion, there exist two drawbacks in most existing methodologies that difficult the applicability of this approach:

On the one hand, most of them, general or protocol-centric, agree on using abstract messages and sequence diagrams to describe protocols [2, 8, 11, 18]. Although these messages represent a high level view of a protocol, which shall be refined later, the tasks that are performed are formulated as a set of messages. This representation implies that the abstraction level falls dramatically since a task requires several messages to be represented. For instance, an information request between two agents must be represented with two messages at least (one to ask, and another to reply). This introduces a semantic gap between tasks and its internal design since it is difficult to identify the tasks represented in a sequence of messages. This representation becomes an important problem regarding readability and manageability of large MAS.

On the other hand, abstractions of protocols (interactions) that allow designers to encapsulate pieces of a protocol that is executed by an arbitrary number of agents have been proved adequate in this context [2, 3, 11, 12, 19]. Unfortunately, interactions are generally used to hide unnecessary details about some views of the protocol. This improves readability and promotes reusability of protocol patterns, but they are not used for bridging the existing semantic gap between tasks and its representation.

1.2 Contributions

We present a different approach to use interactions, which is based on the ideas presented in [3, 19]. As we sketched in Figure 1, our goal is to bridge the gap using interactions to model the tasks to be performed, and Finite State Automata (FSA) to model how to sequence them (see static and dynamic interaction views in Figure 2, and Figure 3). Afterwards, we refine them systematically into simpler ones when it is possible (see Figure 1). This decreases the level of abstraction of complex tasks so that the interactions we obtain are simpler. Thus, they are described internally as message sequences easily, e.g. using AUML [2], which is the last step in our approach.

We have used an abstraction called multi-role interaction (mRI), which was proposed in [14]. An mRI encapsulates a set of messages between an arbitrary number of agent roles. Furthermore, the refinement process we use is based on the ideas presented in [5] since the interaction we use is similar to such used in this work. The refinement process relies on analysing the knowledge used by each role in an mRI and using this information to transform an mRI into several simpler mRIs. An mRI is simpler when both the number of participant roles and the computation made by it decreases. The main advantages of refining mRIs are the followings: First, its internal description is easier since the computation to perform in the obtained tasks is simpler. Second, it is easier to implement interactions with a low number of participants [1, 6]. Finally, mRIs are critical deadlock free regions and they are mutually exclusive. Thus, if the number of participant roles increases, the concurrency grain decreases, what is clearly not desirable[16].

The main drawback of such refinements is that they may lead to deadlocks. In this paper, we also propose a technique to detect if a refinement may introduce deadlocks (see Figure 1); it also characterises them by means of regular expressions that help finding the refinements that are not adequate in a given context. It is based on analysing the FSA that represents the protocol and some previous work on deadlock detection in the context of client/server interactions [4, 7, 17]. It improves on other results in that it can be automated because it does not require any knowledge about the implied, intuitive semantics of the interactions as other approaches.

This paper is organised as follows: in Section 2 we present the related work; in Section 3 we present our ideas on protocol modeling and we show the refinement techniques; in Section 4 we present our approach to the automatic deadlock detection process. Finally, in Section 5, we show our main conclusions.

2. RELATED WORK

As we showed in the previous section, we think that most approaches model protocols at low level of abstraction since they require the designer to model complex cooperations as message-based protocols. This issue has been identified in the Gaia Methodology [19], and also in the work of Caire *et. al.* [3], where the protocol description process starts with a high level view based on describing tasks as complex communication primitives (hereafter interactions). We think that the ideas presented in both papers are adequate for this kind of systems where interactions are more important than in object-oriented programming.

On the one hand, in the Gaia methodology, protocols are modeled using abstract textual templates. Each template represents an interaction or task to be performed between an arbitrary number of participants. Furthermore, interactions are decorated with the knowledge they process and the permissions each role has, their purpose, their inputs and outputs, and so on.

On the other hand, in [3], the authors propose a methodology in which the first protocol view is a static view of the interactions in a system. Each interaction is used by a set of agent roles and they are decorated with the knowledge each role uses/supplies. Later, the internals of these interactions are described using AUML [2].

As the methodologies cited above, we also use interactions to deal with the first stage of protocol modeling. Furthermore, we also represent a static view of interactions and the

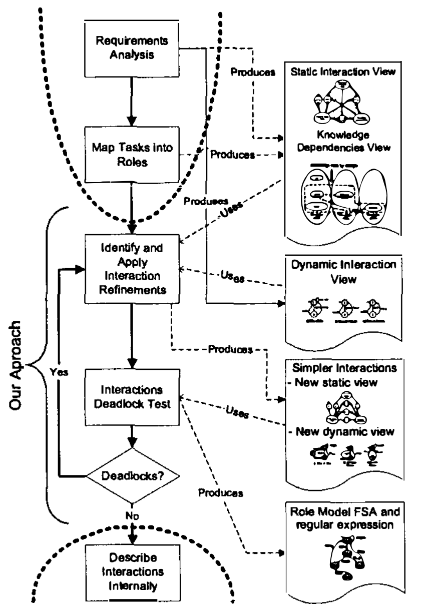

Figure 1: A workflow describing our approach.

knowledge that each role consumes and produces in each of them. Unfortunately, both methodologies do not provide a method for refining complex interactions into smaller interactions that are closer to the implementation level. In this paper, we elaborate on such a method.

The need for interactions has also been identified in other areas such as distributed systems [6, 13]. In this context such interactions have been studied for long, and there exist advanced techniques to refine them (synchrony loosening refinements [6]). The technique that focus on deadlock detection of refined systems was presented in [6]. It is based on designing a formal proof system (*cooperating proof*) that allows to prove a sufficient condition that ensures that a system is deadlock free. Unfortunately, this technique is quite difficult to apply in practice because it requires in-depth knowledge of the implied, intuitive meaning of the interactions, and no automatic proof rules were designed for showing the satisfaction of the sufficient condition.

Our proposal can detect if a refinement may lead to a deadlock situation automatically, and also characterises the set of traces that lead to it by means of regular expressions. It is based on FSA analysis used by many researchers in the context of client/server deadlock detection of interaction models [4, 7, 17].

3. MODELING THE PROTOCOL

In this section we use an example to illustrate our approach. The example we use is a debit–card system, which can be viewed as one of the basic coordination patterns in the agent e-commerce world. It involves three different agent roles (hereafter roles): a point of sales role (PS) that interacts with the user, a customer account manager role(CA), and a merchant account manager role (MA). When a customer uses his or her debit–card, the agent playing role PS agrees with a CA agent and merchant account agent on performing a sequence of tasks to transfer the money

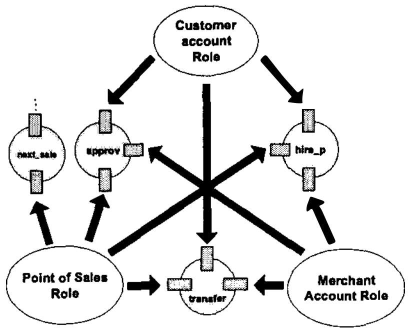

Figure 2: Static interaction view of the example.

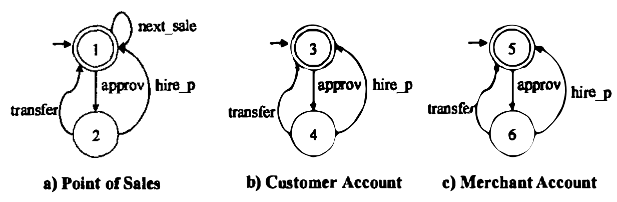

Figure 3: Dynamic interaction view of the example.

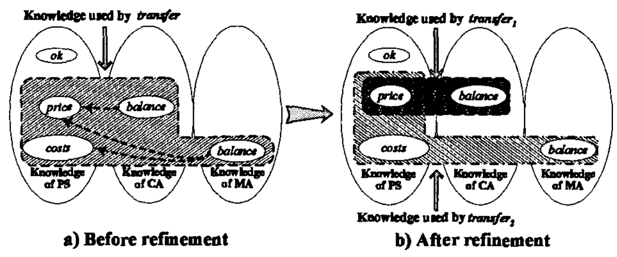

Figure 4: Decoupling mRI $transfer$.

from the customer account to the merchant account, which shall also be charged the costs of the transaction. If the customer account cannot afford the purchase because it has not enough money, the customer account agent then pays on hire–purchase.

As we showed in Figure 1, our approach starts when the tasks to be performed and their mapping onto roles have been already obtained [3, 10, 18]. Then, we model each task as an mRI as we show in the static interaction view (see Figure 2). We model also the knowledge dependencies for each mRI of our example with the information obtained in the requirements analysis stage (see Figure 4a).

The tasks in our example are modeled as the following mRIs: *approv* is used by the CA role to inform the other parties if it can afford a purchase; $transfer$ is used to transfer money from the CA to the MA by means of the PS; mRI *hire_p* is used to buy on hire-purchase; finally, there is a two-party mRI called *next_sale*, which encapsulates the operations needed to read the sum to be transferred and the customer data from his or her debit card.

Once the mRIs are identified and mapped onto roles we represent their possible sequences by means of FSAs (see Figure 3). When an mRI is executed by more than one role it must appear a transition in all its participant roles. Each of these transitions represents the part of the mRIs that a role performs. Whereby, to execute an mRI we must transit from one state to another in all the roles that participate on it. Furthermore, with the algorithms presented in [14], which we outline in section 4, we can automatically infer a single FSA that represents the role model protocol as a whole. This alternative representation can be used for better readability.

Finally, each mRI have to be decorated with some additional information: such as the dependencies between they knowledge it process, a guard for each role, and so on. The knowledge dependencies, as we show in the next section, can be analysed in order to refine mRIs. Furthermore, the guard of mRIs allows each role to decide if it want to execute the mRI or not, which has been proved adequate to deal with proactivity of agents [11, 14].

The model presented before takes advantage of complex mRIs, which provides a high level design. However, it should be refined in an attempt to transform its mRIs into a set of simpler ones that are closer to message sequences description. That is to say, describing them internally shall be easier. This is the next step in our approach.

The refinements are based on analysing the dependencies between the knowledge that roles use from others in a particular mRI. In order to identify which refinements are applicable the designer has to build a dependency graph (see Figure 4a) which shall be analysed with the algorithms proposed in [5, 9].

For example, we can apply a refinement called *decoupling* [6]. It can transform certain n–party mRIs into an m–party mRI ($m < n$) followed by an mRI with $n-m+1$ participants. We can illustrate it by means of mRI $transfer$. Figure 4a shows a diagram in which we have depicted the knowledge of its roles and their dependencies. As shown, both the MA and CA need to update their balances according to some information in the knowledge of the PS. The idea is thus to decouple mRI $transfer$ into two binary mRIs so that the CA updates its balance before the MA. Thus, as we can see in Figure 4b mRI $transfer_1$ shall be executed by PS and CA, and $transfer_2$ by PS and MA. We have applied this refinement to the mRI *hire_p* as well and *participant elimination* [6] to mRI *approv* (see Figure 5 for the new sequences of execution). Others refinements can be found in [6].

The resulting FSAs after applying all refinements are presented in Figure 5. Apparently, they works well but we can discover that the refinements have introduced a deadlock if we take a closer look. Consider a trace in which the following mRIs are executed: *next_sale*, *approv*, $transfer_1$, and $hire\_p_1$. This execution deadlocks because of an unfortunate interleaving in which, after approving a sale and charging the CA, this role is ready to interact with the PS by means of $transfer_2$; however, the MA is readied then to execute both $transfer_1$ and $hire\_p_1$. If $hire\_p_1$ is executed, it leads to a situation in which no role can continue because PS is readying $transfer_2$ and waits for the CA to ready it, the CA is readying *approv* and waits for the PS to ready it, and the MA is waiting for any of them to ready $transfer_1$ or $hire\_p_1$. This situation can be avoided if we use a guard for $transfer_i$ and $hire\_p_i$, which ensures that when one of these mRI is executed, the guard of the others shall be evaluated

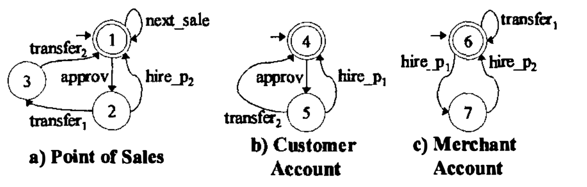

Figure 5: Sequences of mRIs after refinement.

as false. Unfortunately, this is not possible in general.

These refinements allow us to execute several mRIs concurrently since the knowledge they computed before is now computed separately in different mRIs. In addition, they decrease the number of participants of mRIs, which lead us to easier implementations (the protocol to coordinate n parties is more difficult than such for two parties) [1, 6]. Finally, the complexity of the knowledge processed in each mRI decreases thus easing their internal design. For instance, the mRI *transfer* has been broken into two simpler mRIs: $transfer_1$ and $transfer_2$. $transfer_1$ computes the balance of the CA and $transfer_2$ computes the balance of the MA. Thus, simpler computations are performed. Furthermore, the original mRI had three participant roles, and the new mRIs have only two, whose coordination/negotiation protocol is simpler to implement.

4. ENSURING DEADLOCK FREE PROTOCOLS

Our approach to detect deadlocks is based on building an FSA and analysing its paths. Next, we present some results we need, and then we show how to construct the FSA and how to analyse it.

As we can see in Figure 5, the definition of the protocol of each role is done by means of FSAs. They can be characterised as follows:

DEFINITION 1 (FSA). *A finite state automaton (FSA) is a tuple of the form $(S, \Sigma, \delta, s^0, F)$, where S is a set of states, Σ is a set of mRIs (the vocabulary in FSA theory), $\delta : S \times \Sigma \longrightarrow S$ is a transition function that represents an mRI execution, $s^0 \in S$ is an initial state, and $F \subseteq S$ is a set of final states.*

Thus, let $A_i = (S_i, \Sigma_i, \delta_i, s_i^0, F_i)$ $(i = 1, 2, \cdots, n)$ be the set of FSAs that represents each role in a role model. Then, we can build a new FSA $C = (S, \Sigma, \delta, s_0, F)$ that represents the protocol as a whole, where

- $S = S_1 \times \cdots \times S_n$
- $\Sigma = \bigcup_{i=1}^n \Sigma_i$
- $\delta(a, \{s_1, \ldots, s_n\}) = \{s_1', \ldots, s_n'\}$ iff $\forall\, i \in [1..n] \cdot (a \notin \Sigma_i \wedge s_i = s_i') \vee (a \in \Sigma_i \wedge \delta(a, s_i) = s_i')$
- $s_0 = \{s_1^0, \ldots, s_n^0\}$
- $F = \{F_1, \ldots, F_n\}$

This algorithm has been presented in [14] and builds the new FSA exploring all the feasible executions of mRIs. Their states are computed as the cartesian product of all states. Then, for each new state (composed of one state of each role) we check if an mRI may be executed (all their roles can do it from that state); if so, we add it to the result. The FSA that we obtain in our example is shown in Figure 6.

4.1 Analysing the Resulting FSA

The final step consists in analysing the resulting FSA by searching for deadlock states, i.e., states from which a final state cannot be reached.

We use a transition relation called $\longrightarrow_B$ to calculate these states. It is applied on tuples of the form (C, N, X), where C denotes an FSA, N denotes the set of states to be analysed, and X denotes the set of deadlock states found so far. We formalise $\longrightarrow_B$ by means of the following inference rule:

$$\frac{s \in N \wedge s \notin X \wedge P = \mathsf{pred}(s, C)}{(C, N, X) \longrightarrow_B (C, N \setminus P, X \cup P)}$$

Where the predicate pred is defined as follows:

DEFINITION 2 (PREDECESSORS). *Let A be an FSA and $s \in S$ a state. We denote its set of predecessors by $\mathsf{pred}(s, A)$ and define it as follows:*

$$\mathsf{pred}(s, A) = \{s' \in S \mid \exists \sigma \in \Sigma \cdot \delta(s', \sigma) = s\}$$

This transition relation allows us to explore the set of states of an FSA starting at its final states and going back to its predecessors until no new unexplored state is found. The set of unexplored states at that step is the set of deadlock states because there is no path in the FSA that links them to a final state. Therefore, we can define a function deadlock that maps an FSA into its set of deadlock states as follows:

$$\mathsf{deadlock}(C) = C_S \setminus N \text{ if } N \subseteq C_S \wedge$$

$$X \subseteq C_S \wedge (C, C_F, \emptyset) \longrightarrow_B^! (C, N, X)$$

Here, $\longrightarrow_B^!$ denotes the normalisation of $\longrightarrow_B$, i.e., its repeated application to a given tuple until it can not be further applied to the result. If deadlock returns an empty set, then the refinements we have applied do not introduce any deadlocks. Otherwise, we need to characterise the execution paths that may lead to them.

Consider that $\mathsf{deadlock}(C) = \{b_1, b_2, \ldots, b_k\}$, thus, we can build a new set of FSAs $B_i = (C_S, C_\Sigma, C_\delta, C_{s^0}, \{b_i\})$ $(i = 1, 2, \ldots, k)$. Notice that these FSAs have only a final state that is a deadlock state in the original FSA. Thus, if we use the algorithms presented in [7] for transforming an FSA into its corresponding regular expression, we can obtain the set of regular expressions that characterise the execution paths that lead to deadlocks.

If we analyse the FSA in Figure 6, we can easily check that its set of deadlock states is a singleton of the form $\{(3, 4, 7)\}$. Thus, if we make this the only final state, we can obtain the following regular expression that characterises the execution paths that lead to deadlocks: $(next\_sale \mid approv \cdot transf_1 \cdot transf_2 \mid approv{\cdot}hire\_p_1{\cdot}hire\_p_2)^*{\cdot}approv{\cdot}transf_1{\cdot}hire\_p_1$.

Thus, when a set of refinements are applied we can use the technique presented above to search for deadlocks, and if they appear, we characterise it by the deadlock regular expression. Then, we can use this characterization to apply a different set of refinements and repeat this process until

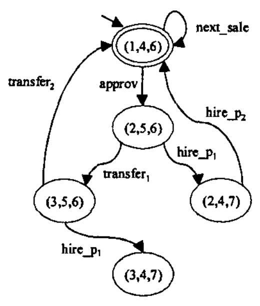

Figure 6: Resulting FSA.

getting a deadlock free protocol. Finally, we obtain a set of new simpler mRIs that can be described internally and implemented easier. In our example the deadlock appears between mRI $transfer$ and $hire_p$. It can be easily avoided not refining one of them, applying another set of refinements, or adding an appropriate guard.

5. CONCLUSIONS

The description of the protocols in a complex MAS may be a difficult, tedious process due to the large number of complex tasks that agents must perform. Thus, in order to palliate this problem, we have proposed a methodology that is based on an interdisciplinary technique that builds on MAS and distributed systems ideas.

Our technique improves previous research in that we add some protocol views between requirements analysis and the description of a protocol by means of message sequences. In these views mRIs are used as first class modeling elements. Furthermore, these mRIs can usually be refined in order to ease its internal description as message sequences. Thus, we provide a progressive method to proceed from requirements analysis to message sequences descriptions. Furthermore, we provide an automatic method to detect deadlocks.

6. REFERENCES

[1] R. Bagrodia. Synchronization of asynchronous processes in CSP. *Transactions on Programming Languages and Systems*, 11(4):585–597, Oct. 1989.

[2] B. Bauer, J. Muller, and J. Odell. Agent UML: A Formalism for Specifying Multiagent Interaction. In *Proc. of 22nd International Conference on Software Engineering (ISCE)*, LNCS, p. 91–103, Berlin, 2001. Springer–Verlag.

[3] G. Caire, F. Leal, P. Chainho *et. al.* Agent Oriented Analysis Using MESSAGE/UML. In *Proc. of AOSE'01*, p. 101–108, Montreal, 2001.

[4] J. C. Corbett. Evaluating Deadlock Detection Methods for Concurrent Software. *IEEE Transactions on Software Engineering*, 22(3):161–180, 1996.

[5] N. Francez and I. Forman. Synchrony Loosening Transformations for Interacting Processes. In J. Baeten and J. Klop, editors, *Proc. of Concurr'91*, 527 in LNCS, p. 27–30, Amsterdam, The Netherlands, 1991. Springer–Verlag.

[6] N. Francez and I. R. Forman. *Interacting Processes.* Addison–Wesley, 1996.

[7] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[8] C. Iglesias, M. Garrijo, and J. Gonzalez. A Survey of Agent-Oriented Methodologies. In J. Müller, M. P. Singh, and A. S. Rao, editors, *Proc. of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555, p. 317–330. Springer-Verlag: Heidelberg, Germany, 1999.

[9] S. Katz, I. Forman, and W. Evangelist. Language Constructs for Distributed Systems. In *IFIP TC2 Working Conference on Programming Concepts and Methods*, Galilea, Israel, 1990.

[10] E. Kendall, U. Palanivelan, and S. Kalikivayi. Capturing and Structuring Goals: Analysis Patterns. In *Proc. of the 3rd European Conference on Pattern Languages of Programming and Computing*, 1998.

[11] J. Koning, M.Huget, J. Wei, and X. Wang. Extended Modeling Languages for Interaction Protocol Design. In M. Wooldridge, P. Ciancarini, and G. Weiss, editors, *Proc. of 2nd Internationa Workshop on Agent-Oriented Software Engineering (AOSE'02)*, LNCS, Montreal, Canada, May, 2001. Springer–Verlag.

[12] H. J. Levesque, P. R. Cohen, and J. H. T. Nunez. On Acting Together. In T. Dietterich and W. Swartout, editors, *Proc. of the 8th National Conference on Artificial Intelligence (AAAI-90). Boston, MA, USA.*, p. 94–99. AAAI Press, 1990.

[13] G. Papadopoulos and F. Arbab. Coordination Models and Languages. In *Advances in Computers*, volume 46. Academic Press, 1998.

[14] J. Peña, R. Corchuelo, and J. L. Arjona. Towards Interaction Protocol Operations for Large Multi-Agent Systems. In J. Rash et al. editors, *Proc. of the Second International Workshop on Formal Approaches to Agent-Based Systems*, LNAI, page To be pubblished, NASA-Goddard Space Flight Center, Greenbelt, MD, USA, 2002. Springer–Verlag.

[15] R. Pressman. *Software Engineering: a Practitioner's Approach.* MacGraw Hill, New York, N.Y., 1986.

[16] M. Singhal. Deadlock detection in distributed systems. *Computer Magazine of the Computer Group News of the IEEE*, 22(11):37–48, 1989.

[17] M. Y. Vardi and P. Wolper. An Automata-Theoretic Approach to Automatic Program Verification. In *Proc. 1st Annual IEEE Symp. on Logic in Computer Science, LICS'86, Cambridge, MA, USA, 16–18 1986*, p. 332–344. IEEE Computer Society Press, Washington, DC, 1986.

[18] M. Wood and S. A. DeLoach. An Overview of the Multiagent Systems Engineering Methodology. In *Proc. of the 1st International Workshop on Agent-Oriented Software Engineering*, number 1957 in LCNS, Limerick, Ireland, 2001. Springer–Verlag.

[19] M. Wooldridge, N. R. Jennings, and D. Kinny. The GAIA Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.