Using semantic causality graphs to validate MAS models

Guillermo Vigueras[1], Jorge J. Gómez[2], Juan A. Botía[1] and Juan Pavón[2]

[1]Facultad de Informática
Universidad de Murcia
Spain

[2]Facultad de Informática
Universidad Complutense de Madrid
Spain

Abstract Multi-agent systems methodologies define tasks and tools covering all the development cycle. Modelling and implementation activities are well supported and experimented. However, there has been few effort invested in debugging and validation techniques for MAS compared to modelling and implementation stages. This paper addresses the lack of tools to debug and validate a MAS. We propose a visual tool to debug and validate some multi-agent system specification, capturing causality among messages sent by agents and showing semantic information related to interaction protocols used by agents.

Keywords: Intelligent agent-based systems, MAS debug, MAS validation.

1 Introduction

Modelling multi-agent systems is a delicate task prone to errors. Most of the times, it is a problem of lack of specialised verification and validation tools. One of design parts that developers should specially take care of, is interaction protocols specification. By means of these protocols, agents communicate to achieve their individual or common goals. So analyzing interactions provides valuable information to determine agent behavior. Agents methodologies offer to developers tasks and activities, along MAS development cycle, to model multi-agent systems. However the number of debug and validation tools offered by these methodologies is not comparable as the number of development tools offered. Visual tools that represent an abstract view of the system can help developers to perform debug and validation tasks. In this way causality is and important concept from distributed systems that can be adapted to multi-agent systems. Causality of a concrete event (i.e. a message exchanged, an undesired result of the interaction studied, an unexpected conversation, etc.), in the context of multi-agent interactions, may be loosely defined as the cause which directly or indirectly leaded to generate the event. What we propose in this paper is adding, to causality graphs proposed in [6], semantic information specified inside some interaction protocol. Semantic information defines how received and sent messages affect to agents, i.e. what actions are executed and how agents mental state are modified by those actions. In order to add semantic information, we propose to use the meta-models defined in INGENIAS methodology [5] and the information provided by those meta-models about the MAS. Using this information, developer will be able to perform a better diagnose and find the reason of possible error in the agents behavior more quickly.

The rest of the paper is organized as follows. Section 2 introduces the notion of causality in the context of a MAS interaction. Section 3 describes INGENIAS methodology. Section 4 describes a MAS example modelled using INGENIAS and how semantic causality graphs can help developer to debug and verify it. Section 5 puts our causality graph in the appropriate context and finally, section 6 enumerates most important conclusions and open issues.

2 Representing causality through MAS interactions

In order to integrate debug, validation and verification in the MAS development cycle, it is important to offer tools performing these tasks in an easy way. This kind of tools should offer an abstract view of a MAS where errors can be detected in an easy and fast way. The paper proposes a tool that uses causality to guide the developer towards the root of the problem. To illustrate the role of causality, a brief example follows. There is a MAS in which an agent (denoted by Initiator) wants to search bibliographic references about some subject. To perform this task the Initiator communicate with another agent (denoted by Broker) using the FIPA Brokering interaction protocol, and delegate on the Broker to do the search task. The `proxy` message sent by Initiator agent, contains the following information: a referential expression denoting the target agents to which the Broker should forward the requested task and the communicative act to forward. After notify to Initiator that the petition is accepted, the broker agent forward again the search request, using the communicative act (`request`) and the target agent list, received both from Initiator. The target agent list contains two agents, denoted by Searcher1 and Searcher2, respectively, so Broker communicate with Searcher1 and Searcher2 using FIPA Request interaction protocol . Searcher1, communicate with agents BookSeller1, BookSeller2, and BookSeller3 representing each one some e-library, to perform the search, using the FIPA Request interaction protocol. Searcher2, is not able to understand the `request`, so it doesn't send some message. After, some interactions, Searcher1 receives only failure messages, which are forwarded to Broker, which forward again the failure to Initiator. The example described above is shown in figure 1.

After MAS execution, the developer realizes that Initiator's search returns failure, but the source of the error is not evident. Looking at the graph shown in figure 1, the developer discovers that Initiator's search has failed because Broker returns failure too. Looking at the bottom of the graph, it seems Broker behavior is correct, because all messages received are failures. Tracking back agents interaction, developer realize that all responses received by Searcher1, are failures too, and Searcher2 does not do anything. The MAS programmer, reviews agents code and finds which databases are been accessed by agents.

Once databases are found, the programmer checks whether the bibliographic reference searched by Initiator agent, exits or not in e-libraries that are accessed by BookSeller1, BookSeller2 and BookSeller3. He discovers that the searched reference does not exist and realizes that the problem is in the target agent list elaborated by Initiator. Probably, it was caused by wrong information about the environment, maybe provided or modified by some information source. Related with Searcher2, developer, after reviewing agent's code, realizes that this agent is not able to process a `request` message, so the other cause error is found too. In this example, causality graph is useful for developer to find error causes, but he has to inspect agents code, inspection that implies consume more development time.

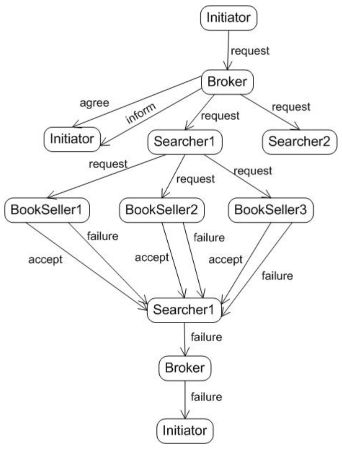

Figure 1. Example in which agents search bibliographic references.

The graph shown in figure 1, can help to developer to discover which agent has a wrong behaviour, respect to what is expected. But once the agent is found the developer can't know which is the internal agent state that cause the wrong behavior. We propose in this paper to add semantic information, extracted from agents interaction protocols, to the graph shown in figure 1 to increase MAS execution information and reduce, in this way, debug and validation time.

3 The INGENIAS Methodology

INGENIAS defines process, methods and tools for developing MAS, from the more abstract analysis, with use cases and system goals, to implementation of code. In order to cope with the complexity of MAS specification, each MAS is considered from five complementary viewpoints [5]:

– Agent viewpoint. This viewpoint is concerned with the functionality of each agent: responsibilities (what goals an agent is compromised to pursue) and capabilities (what tasks is able to perform). The behaviour of the agent is defined through three components: (1) the mental state (an aggregation of mental entities such as goals, believes, facts, compromises, and others), (2) the mental state manager, which provides for operations to create, destroy and modify mental entities, and (3) the mental state processor (which determines how the mental state evolves and it is described in terms of rules, planning, etc.).

- Organisation viewpoint. The organisation describes the framework where agents, resources, tasks and goals coexist. This viewpoint is defined by the organisation structure (with group decompositions), social relationships (power relationships mainly), and functionality. The functionality of the organisation is defined by its purpose and tasks.
- Task/Goal viewpoint. It considers the relationships among goals and tasks or plans, and describes the consequences of performing a task, and why it should be performed (i.e. it justifies the execution of tasks as a way to satisfy goals). For each task, it determines which elements are required and what outputs are expected. To identify which goals are influenced by a task execution, the methodology defines satisfaction and failure relationships. Finally, the tasks/goals model can detail how a solved goal affects other existing goals by using decomposition and dependency relationships. It is useful to know that by solving a sub-goal, a super-goal can be solved too.
- Interaction viewpoint. The interaction viewpoint addresses the exchange of information between agents, or between agents and human users. The definition of an interaction requires the identification of: actors in the interaction (roles or agents), interaction specification (the protocols, the notation for the protocols, the interleaving of task execution during interaction, the information exchanged during the interaction), context of the interaction (goals pursued, mental state of the participants before, during, and after the interaction), and nature of the interaction (coordination, negotiation, planning, etc.).
- Environment viewpoint. It defines the entities with which the MAS interacts. These entities are resources (elements required by tasks that do not provide a concrete API, like CPU, file descriptors, or memory), applications (they offer some local or remote API), or other agents (from other organisations). In the diagrams used to define this particular viewpoint, agents are associated to resources or applications. Also, the perception relationships are established, defining the sensors of the agent, and actuators, by making an agent responsible of a resource or an application.

INGENIAS specifications are produced with an editor supporting the creation of diagrams for these viewpoints, the INGENIAS Development Kit (IDK). From those diagrams the IDK is able to automatic generate code to execute the modelled MAS. Moreover the IDK offers a framework for the implementation of modules that allow to browse viewpoints described above. Combining both the generation code and viewpoints browser utilities it is suitable to build tools to debug and validate the MAS specification. An example of those tools is the one explained in the following section.

4 Debugging an Example INGENIAS MAS Model

This section shows an example of how semantic causality graphs can aid developer to get more information about a running MAS under development. For this purpose, this section presents an example MAS specification modelled with INGENIAS. Later, it is shown how debug and validation tasks are performed using semantic causality graphs. Taking account that the goal is extending causality graphs that appear in figure 1, to show how agents interactions affect to agent internal state, different INGENIAS viewpoints has to be browsed. In the next section it is described the MAS specification used and which INGENIAS meta-models are used.

4.1 MAS Example specification description

The system intends to provide a ticket selling service where a user representative can contact each cinema and proceed with the transaction. The case study considers a user connecting to the system and instructing a representative (User assistant) to obtain a cinema ticket. The assistant contacts another representative (buyer) so that this representative obtains the ticket. The representative contacts different cinemas representatives (seller) until an adequate cinema is located (adequate means there are free seats and the price is good enough). Once obtained, the ticket is delivered, through the different representatives, to the user.

Taking into account that semantic information associated to agents interactions is the main concern, the INGENIAS Interaction viewpoint and Task/Goal viewpoint is consulted. The first one gives information about which tasks are executed by one agent when receives a message. Focusing on these tasks is necessary because modifications performed by some task when a message is received will influence the next message. Related to the interaction viewpoint, the Task/Goal viewpoint informs of the modifications to perform over an agent mental state, those executed by enacted tasks. In the system described above there are two interactions. The first one between User assistant agent and Buyer agent, in which User assistant instructs Buyer to get a cinema ticket. The second one between Buyer and Seller agent negotiates the buy of the cinema ticket required. In figure 2, it is shown a diagram from Interaction viewpoint for specifying the interaction protocol used by buyer agent and seller agent, to negotiate cinema ticket required. On the right, it appears a detailed specification of the task *Process offer* from Task/Goal viewpoint. In the interaction protocol specification, messages sent by agents are represented through *Interaction units*, each one containing one speech act or performative. When a message is sent by one agent (or role) this is specified by means of relation *UIInitiates*. When a message is received, it is represented by means of relation *UIColaborates*. Semantic information is defined associating one task to a sent or received message event. For each task into Task/Goal viewpoint it is identified entities from agent mental state being consumed/produced, and environment entities being used. Due to space limitations, only one task is shown. As the reader can see, the task defined in figure 2, satisfies the goal *GetTicket* (relation *Satisfaction*) and the role *Buyer* is responsible of execute the task (relation *WFResponsable*). In addition, the task will produce the fact *Pay the money* if the buyer agent is interested in the ticket offered, and will produce the fact *Not interested* if isn't interested. Taking account this semantic information, in the next section it is shown an example of semantic causality graph.

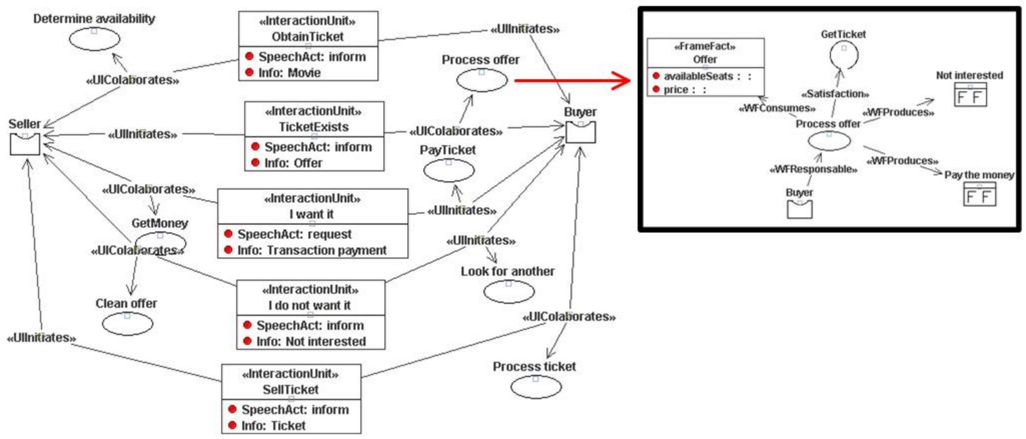

Figure 2. Definition of interaction protocol to buy a cinema ticket, in which interaction semantic information is added. On the right the task *Process offer* is defined too

4.2 Validating the example using semantic causality graphs

There is a system in which there is one user and his user assistant agent , one Buyer, and two sellers (Seller1 and Seller2) representing one cinema each one. Let's suppose that the user instructs the User assistant to obtain a cinema ticket. The User assistant starts an interaction with Buyer agent, to get the cinema ticket. Later on, Buyer agent starts another interaction with Seller1 to buy the required ticket. The ticket offered is refused by Buyer so it starts a new interaction with Seller2 and again the offer is refused. The developer, looking at the resultant execution, begins to think about the reasons for the refused offers. The graph shown in figure 3 allows the developer to know the reasons that lead to Buyer to refuse tickets offered, getting semantic information from the running MAS (shown inside black rectangles in figure 3), and without necessity to inspect agents code. By means of semantic information, the developer can realize that the tickets are refused due to the offer sent by each seller to Buyer agent (see facts *Offer* produced by Seller1 and Seller2 in figure 3). In both cases the price of the offer is very high, as is shown in the attribute *Price* of offers from Seller1 and Seller2. To solve the problem, by means of semantic information added, the developer realizes that the code of the task *Determine availability*, that is in charge of elaborate the refused offer, should be reviewed.

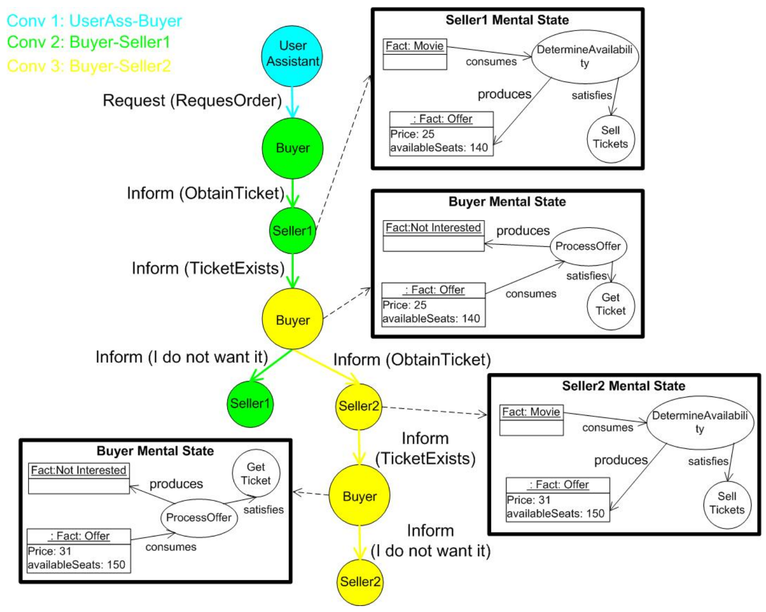

Figure 3. Example in which agents search bibliographic references.

5 Related works

There are previous works on using causality and visual debugging for MAS, and specially the later ones. Causality graphs are used extending them to capture causality among different MAS events. in [3]. In that work, a developer defines agent concepts and causal relation among those pre-defined concepts is shown (e.g. a believe created as a consequence of a received message). The work [3] is complementary to the one presented, because the causality graph is at intra-agent level and not at inter-agent level.

An example of visualization tool to assist in testing and debugging MAS is the one described in [2]. Simple communication graphs are used there, i.e. nodes are agents and arcs exits between nodes if they exchanged any message. However, when these communication graphs get so complex to be visualized (i.e.

many agents and many arcs), a simplification process based on clustering is used to group agents. Different criteria are used to decide on the grouping.

Keeping the attention on visualization tools for MAS, other interesting example is the one presented in [4]. In this case, the developer can use different kinds of visual diagrams. For example, Gantt based diagrams show decomposition, dependency and ordering among agents tasks. Using 3D graphical representation, in [1] is proposed a tool to debug some MAS. This example follows a sequence diagram model and allow to filter events occurred in the MAS, by means of special queries, depending on user interests.

6 Conclusions

This paper has proposed a visual representation of MAS behavior aimed to aid the developer in MAS debugging, verification and validation tasks. This visual representation allows a deeper and quicker understanding of the system behavior. The deeper understanding is provided by means of causality concept, which allows to know the cause message of a given message, and through semantic information, which shows agents mental state that leads to send a message to each agent.

We keep working on a generic algorithm for producing semantic graphs using as input INGENIAS models. Moreover semantic information can lead to program tools to perform automatic reasoning about MAS behaviour.Error causes could be automatically extracted as much as some suggestions to solve the problem. In addition we plan to develop different methods to simplify the graph, for example to show only both interaction and semantic information about one agent. This can be helpful in complex graphs with a high number of agents.

7 Acknowledgements

This work has been supported by the Spanish Council for Science and Education under grant TIN-2005-08501-C03-02 and the project "Methods and tools for multi-agent systems modelling" supported by Spanish Council for Science and Technology with grant TIN-2005-08501-C03-01.

References

1. S. Ilarri, J.L. Serrano, E. Mena, and R. Trillo. 3D monitoring of distributed multiagent systems. In *International Conference on Web Information Systems and Technologies (WEBIST 07), Barcelona (Spain)*, pages 439–442. INSTICC Press, ISBN 978-972-8865-77-1, March 2007.
2. Juan M. Hernansaez Juan A. Botia and Antonio F. Gomez-Skarmeta. On the application of clustering techniques to support debugging large-scale multi-agent systems. In *Programming Multi-Agent Systems Workshop at the AAMAS*, Hakodate, Japan, 2006.

3. D. N. Lam and K. S. Barber. Comprehending agent software. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 586–593, New York, NY, USA, 2005. ACM Press.
4. Divine T. Ndumu, Hyacinth S. Nwana, Lyndon C. Lee, and Jaron C. Collis. Visualising and debugging distributed multi-agent systems. In ACM Press, editor, *AGENTS '99: Proceedings of the third annual conference on Autonomous Agents*, pages 326–333, 1999.
5. J. Pavon, J. J. Gomez-Sanz, and R. Fuentes. *Agent-Oriented Methodologies*, chapter The INGENIAS Methodology and Tools, pages 236–276. Idea Group Publishing, 2005.
6. Guillermo Vigueras and Juan A. Botia. Tracking causality by visualization of multi-agent interactions using causality graphs. In *Programming Multi-Agent Systems Workshop at the AAMAS (to appear)*, Honolulu, Hawai, 2007.