Supporting Social Knowledge in Multiagent Systems through Event Tracing

Luis Búrdalo, Andrés Terrasa, Ana García-Fornes and Agustín Espinosa

***Abstract*—Social knowledge is one of the key aspects of MAS in order to face complex problems in dynamical environments. However, it is usually incorporated without specific support on behalf of the platform and that does not let agents take all of the advantage of this social knowledge. At present time, the authors of this paper are working in a general tracing system, which could be used by agents in the system to trace other agents' activity and that could be used as an alternative way for agents to perceive their environment. This paper presents first results of this work, consisting of the requirements which should be taken into account when designing such a tracing system.**

***Index Terms*—Agents, multiagent systems, social knowledge, tracing systems.**

I. Introduction

THESE days, the use and importance of multiagent systems (MAS) has increased because their flexible behavior is very useful to deal with complex problems in dynamic and distributed environments. This is not only due to agents individual features (like autonomy, reactivity or reasoning power), but also to their capability to communicate, cooperate and coordinate with other agents in the MAS in order to fulfil their objectives.

The necessary knowledge to support this social behavior is referred to by Mařik et al in [15] as *social knowledge*. This social knowledge plays an important role in increasing the efficiency in highly decentralized MAS. Social abstractions such as teams, norms, social commitments or trust are the key to face complex situations using MAS; however, these social abstractions are mostly incorporated to the MAS at user level; this is, from the multiagent application itself, without specific support from the multiagent platform, by means of messages among agents or blackboard systems. This weak integration of high level social abstractions, also mentioned by Bordini et al in [3], prevents agents in the MAS from exactly knowing what is happening in their environment, since they depend on other agents actively informing them about what they are doing. This dependance on other agents sets out two major problems. First, it can lead to excesive overhead in some of the agents. And second, it is also difficult to trust the information provided directly by other agents using messages in open MAS.

An alternative solution to provide social knowledge could be an event tracing system, integrated within the multiagent platform, which could be used by agents in the system to perceive their environment without having to actively notify each change to the rest of the agents which could be interested in what they do. Such a tracing system, integrated within the multiagent platform and providing a trustworthy event set which were capable to reflect not only communication among agents, but also agents' perceptions, etc, could be used as a way to provide social knowledge to the MAS. Also, such a tracing system would be more trustworthy than agent messages, since the information would not be proportioned by agents, but by the multiagent platform itself. Agents could trust the trace system as much as they can trust the multiagent platform.

Applications which extract information from the system by processing event streams at run time are already considered in the field of event driven architectures [14] and the idea of an standard tracing system available for processes in a system already existed in the field of operating systems (and at present it is contemplated by the POSIX standard[11]). These concepts can be applied to th field of MAS, where event tracing is still considered a facility to help MAS developers in the verification and validation processes.

This paper presents the requirements of such a general, platform-integrated tracing system applied to MAS. These requirements should be taken into account in order to develop a general abstract trace model for MAS, which could be finally incorporated to a real multiagent platform. The rest of the paper presents is structured as folows: Section II comments existing work by other authors in the field of tracing MAS. Section III presents a set of requirements which should be taken into account in order to design a general tracing system which could be used to improve agents sociability. Finally, section IV comments this work's main conclusions and future work which is still to be carried out in order to incorporate such a tracing system to a MAS.

II. Event tracing in multiagent systems

One of the most popular tracing facilities for MAS is the Sniffer Agent provided by JADE[1]. This tool keeps tracking of all of the messages sent or received by agents in the system and allows the user/administrator/developer to examine their content. These messages can be stored in a log file to be examined after the application has stopped running, so that the MAS can also be traced off-line. JADE also provides an Introspector Agent, which can be used to examine the life cycle of any agent in the system, its behaviors and the messages it has sent or received.

JADEX[18] provides a Conversation Center, which allows a user to send messages directly to any agent while it is

All authors are from:
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
cno./ de Vera S/N - 46022 Valencia (SPAIN)
E-mail: {lburdalo,aterrasa,agarcia,aespinos}@dsic.upv.es

executing and to receive answers to those messages from a user-friendly interface. It also provides a DF Browser to track services offered by any agent in the platform at run time and a BDI Tracer which can be used to visualize the internal processes of an agent while it is executing and show causal dependencies among agents' beliefs, goals and plans. Apart from these facilities, JADEX also incorporates a Remote Agent, which provides access to some of JADE's tracing facilities, like the Agent Introspector and the Sniffer Agent.

The JACK[20] multiagent platform does not provide a sniffer agent, but it supports monitoring communication among agents by means of Agent Interaction Diagrams. It also provides other introspecting tools with different functionalities: a Design Tracing Tool, to view internal details of JACK applications during execution, and a Plan Tracing Tool, to display and trace the execution of plans and the events that handle them. JACK also provides debugging tools that work at a lower level of abstraction in order to debug the multiagent system in a more exhaustive way: Audit Logging, Generic Debugging/Agent Debugging.

Other examples of tracing facilities provided by platforms is ZEUS' Society Viewer[10] which, apart from showing organisational inter-relationships among agents in the system, it can also show messages exchanged among agents. ZEUS also provides an Agent Viewer, which allows the user/administrator to monitor and change the internal state of the agent, its actions, used resources, etc. JASON[4], [5] provides its Mind Inspector Tool, to examine the internal state of agents across the distributed system when they are running in debugging mode.

Apart from those tools provided by multiagent platforms themselves, there are many tracing facilities provided by third party developers. This is the case of Java Sniffer[21], developed by Rockwell Automation, a stand alone java application based on JADE's Sniffer Agent which is able to connect to a running JADE system in order to track messages among agents, to reason about them and to show them to the user from different points of view. Another third party tool based on JADE's Sniffer Agent is ACLAnalyser[9], which intercepts messages interchanged by agents during the execution of the application and stores them in a relational database. After the execution, this message database can be inspected to detect social pathologies in the MAS. Later work by the same authors ([8]) combine results obtained with ACLAnalyser with data minning techniques to help in the MAS debugging process.

MAMSY, the management tool presented in [19] lets the system administrator monitorize and manage a MAS running over the Magentix multiagent platform[2]. MAMSY provides graphical tools to interact with the MAS and visualize its internal state at run time, including not only nodes and agents, but also organizational units. It also provides a message tracing tool, similar way to JADE's Sniffer Agent, which lets the system administrator visualize message interchange among agents.

In [16], the authors describe an advanced visualisation tools suite for MAS developed with ZEUS, although the authors also claim these tools could be used with other platforms (more precisely, with CommonKADS). The developed suite allows for inspecting message interchange among agents in a society, displaying graphically the different tasks in the society and its execution state, examining and modifying the internal state of any of the agents in the system and comparing statistics not only for individual agents, but also for agent societies. It also allows for the graphical display of the different tasks in the society and their execution states, examining and modifying the internal state of any of the agents in the system and comparing statistics not only for individual agents, but also for agent societies.

Tracking messages has also been used in [17], which comments an ampliation of the Prometheus methodology and the related design tool to help the designer to detect protocol violations by tracing conversations among agents in the system and to detect plan selection inconsistencies.

Lam et al present in [13] an iterative method based on tracing multiagent applications to help the user understanding the way those applications internally work. Lam et al also present a Tracer Tool which implements the described Tracing Method. The Tracer Tool can be applied to any agent system implementation, regardless of agent or system architecture, providing it is able to interface with Java's logging API (directly or via a CORBA interface). Results obtained with this method were presented in [12]. Bose et al present in [7] a combination of this Tracer Tool with a Temporal Trace Language (TTL) Checker presented in [6]. This TTL Checker enables the automated verification of complex dynamic properties against execution traces.

As it can be appreciated, tracing facilities in MAS are usually conceived as debugging tools to help in the validation and verification processes. It is also usual to use these tracing tools as a help for those users which have to understand how the MAS works. Thus, generated events are destinated to be understood by a human observer who would probably use them to debug or to validate the MAS and tracing facilities are mostly human-oriented in order to let MAS users work in a more efficient and also comfortable way. Some multiagent platforms provide their own tracing facilities, although there is also important work carried out by third party developers. However, even those tracing facilities which were not designed by platform developer teams are usually designed for a specific multiagent platform. There is not a standard, general tracing mechanism which let agents and other entities in the system trace each other as they execute like the one provided by POSIX for processes.

III. Tracing system requirements

From the viewpoint of the tracing process, a MAS can be considered to be formed by set of tracing entities, or components that are susceptible of generating and/or receiving tracing *events*. The tracing system needs to consider, at least, the following list of components inside the MAS as tracing entities: agents, organizational units (or any type of agent aggregation supported by the multiagent platform) and the multiagent platform itself (and its components).

Unlike existing work on tracing MAS, previously mentioned in Section II, a tracing system which could be used as a

knowledge provider must not be human-oriented, but entity-oriented, so that these tracing entities are able to receive events and process them or incorporate them to their reasoning process at run time in order to take advantage from that.

In order to generate trace events, the source code of tracing entities needs to be *instrumented* to include the code which actually produces such events. Attending to where this instrumentation code is placed, trace events can be classified as *platform events* or *application events*.

Platform events are instrumented within the source code of the platform (either in its "core" or in any of its supporting agents). These events represent the generic, application-independent information that the platform designer intends to provide to agents. On the other hand, application events are instrumented within the code of the application agents. These events represent customized run-time information defined by the application designer in order to support specific needs of the application agents.

The rest of the section presents a set of requirements which should be taken into account when developing such a tracing system. These requirements have been classified in three main groups: functional, efficiency and security requirements.

A. Functional requirements

Tracing roles. Any tracing entity in the MAS must be able to play two different roles in the tracing process: *event source* (*ES*) and *event receiver* (*ER*). From the viewpoint of tracing entities, these two tracing roles are dynamic and not exclusive, in the sense that each tracing entity can start and stop playing any of them (or both) at any time, according to its own needs. The relation between ES and ER entities is many to many: it must be possible for events generated by an ES entity to be received by many ER entities, as well as it must also be possible for an ER entity to receive events from multiple ES entities simultaneously.

Chronologically ordered event delivery requirement: Events generated in the system must be delivered to ER entities in chronological order or, at least, include information related to the time when they were produced to allow ER entities to process them in chronological order.

Dynamic definition of event types. Trace events can be classified in event types attending to the information which is generated and attached to them when they are generated. In order to let the event processing be more flexible and efficient, it must be possible for tracing entities to dynamically define new event types at run time. This must be applied to both platform and application event types.

Publication of event types. At any time, ER entities must be able to know which ES entities are producing events and of which types. So, as a consequence of event types being dynamic, the tracing system should keep and up-to-date list of such traceable event types (and ES entities) and to make this list available to all tracing entities in the MAS.

On-line and off-line tracing. In order to let entities work with both historical and run time information, both on-line and off-line tracing should be supported. In on-line tracing, events are delivered to ER entities as they are traced by the tracing system (with a potential delay due to the internal processing of events by the tracing system). In contrast, in off-line tracing, events generated by ES entities are not delivered to running entities, but stored in a log file. Both tracing modes must not be exclusive, meaning that it must be possible for the events generated by any ES entity to be delivered to some ER entities while also being stored in some log files. However, the tracing system does not need to support concurrent access to the events stored in a log file.

B. Efficiency requirements

In any computing system, tracing can be a very expensive process in terms of computational resources. In the case of MAS, the fact that they are by nature highly decentralized systems, both in number of running entities (agents) and hosts, can make their tracing even more expensive. In this context, the tracing process must be optimized in order to minimize the overhead it produces to the system, since a very sophisticated but excessively costly tracing system can become completely useless in practice. The following list introduces a minimal set of efficiency design guidelines that should be considered when designing a tracing system for MAS, in order to make this system realizable and useful. The first two requirements focus on the potential overload of the tracing system while the last one allows entities to set their own limits in the resources devoted to the tracing process.

Selective event delivery. Each ER entity should be able to express which event types it wants to receive, and the tracing system should only deliver events which belong to such types to the entity. Furthermore, each ER entity should be able to change dynamically which events it wants to receive, since entities may need different tracing information at different times during their execution.

Selective event tracing. The tracing system should not spend resources in tracing events which belong to event types that currently no entity wants to receive.

Resource limit control. Each ER entity should be able to limit the maximum amount of its resources to be allocated to receive events, both in on-line and off-line tracing modes. In on-line tracing, if there is some memory data object where events are delivered to until they are retrieved by the corresponding ER entity, then this entity should be able to define the maximum amount of memory devoted to such data object. In off-line tracing, the ER entity that sets the tracing up to the corresponding log file should be able to define the maximum size of the file.

C. Security requirements

Tracing in an open MAS has obvious security issues, since many of the events registered by the tracing system may

contain sensitive information that can be used by agents to take advantage from, or even to damage, the MAS. This scenario enforces the necessity of applying some security policy over the events that can be delivered to entities, specially if they are application entities. This policy can be materialized in many different ways, but in essence, it has to allow for the definition of security rules in the MAS that limit the availability of events to the *right* ER entities. The following list of requirements express a minimum set of restrictions by which it is possible to incorporate such security rules to the tracing system.

Authorization to ER entities. Each ES entity in the system must be able to decide which ER entities can receive the events that it generates. This can be accomplished by means of an authorization mechanism, provided by the tracing system, which can be used by ES entities to restrict the event types that are available to each ER entity. Such authorization rules must be dynamic, so that ES entities are able to modify the list of authorized ER entities corresponding to each event type at run time.

Supervisor entities. Situations where an entity must be able to access to other ES entity's events in order to fulfil its objectives, even though the ES entity does not agree with that, are very common in MAS. This can happen, for instance, in normative environments where an agent has to watch the other in order to verify that norms are not being violated and to apply the corresponding sanctions in case they are. The tracing system must also provide mechanisms to let an ER receive events generated by an ES without its authorization under some circumstances.

Delegation of authorizations. If an ER entity is currently authorized to be delivered events corresponding to certain event types, then this entity can delegate this authorization to other ER entities in the system; then, each of them can do so with other entities (potentially forming an *authorization tree*). At any node in the tree, the corresponding entity can add or remove delegations dynamically. If a delegation is removed, all the potential subsequent delegations (subtree) are also removed.

Platform entities authorization. By definition, the tracing system must be granted the authorization for all event types defined in the MAS, both at the platform and application levels. This is required for the tracing system to be able to keep track of any event being generated in the MAS, independently of the privacy rules defined by each ES entity.

IV. Conclusions and future work

Social knowledge is one of the most important features that make MAS appropiate to deal with complex problems in dynamic and distributed environments. The key to this is the capacity of agents to communicate and coordinate with other agents in the MAS in order to get their objectives. This capacity, though based on high level social concepts such as social commitments, trust, norms or reputation, is usually incorporated to the MAS at user level, using messages or blackboard systems, without support from the multiagent platform. This can produce too much overhead, reducing the scalability of the MAS. Also, it has to be taken into account that sometimes it is difficult to trust information from other agents, specially in open MAS.

A general event tracing system, which agents in the MAS could use to trace other agents in their environment, could be used as a more appropiate and trustworthy social knowledge provider. This paper presents the first step towards defining such a tracing system, which is the identification of its requirements. This paper has identified requirements in different aspects: functionallity, efficiency and security.

Some of the presented requirements set important problems out. Some of these problems are more obvious. For example, the problem of delivering events in chronological order in a distributed MAS. However, others are less evident. For instance, the problem of determining which ES entity is the owner of each trace event, since the instrumented code that produces an event is not always within the source code of the entity which originated it. Just as an example, consider events could as property of those entities which source code has been instrumented to produce them. In this case, all platform events would belong to the multiagent platform, while agents in the MAS would only be owners of application events. It could be more understandable and easier to incorporate considering that events belong to the ES entity which originated them.

Future work will include the design of a general abstract model for MAS which contemplated all of the requirements exposed above and which, after that, could be implemented and incorporated to a multiagent platform.

Acknowledgment

This work is partially supported by projects PROMETEO/2008/051, CSD2007-022 and TIN2008-04446, which is co-funded by the Spanish government and FEDER funds.

References

[1] U. AGREEMENT. Jade administrator's guide. *sharon.cselt.it*.

[2] J. Alberola, L. Mulet, J. Such, A. García-Fornes, A. Espinosa, and V. Botti. Operating system aware multiagent platform design. *Fifth European Workshop On Multi-Agent Systems (EUMAS 2007)*, pages 658–667, 2007.

[3] R. Bordini, M. Dastani, and M. Winikoff. Current issues in multi-agent systems development (invited paper). *Post-proceedings of the Seventh Annual International ...*, Jan 2007.

[4] R. Bordini and J. Hübner. Jason: A java-based interpreter for an extended version of agentspeak. page 31, Mar 2007.

[5] R. Bordini, J. Hubner, and R. Vieira. Jason and the golden fleece of agent-oriented programming. *MULTIAGENT SYSTEMS ARTIFICIAL SOCIETIES AND SIMULATED ...*, Jan 2005.

[6] T. Bosse, C. Jonker, L. van der Meij, and A. S. .... Specification and verification of dynamics in cognitive agent models. *... of the sixth international conference on intelligent agent ...*, Jan 2006.

[7] T. Bosse, D. Lam, and K. Barber. Tools for analyzing intelligent agent systems. *Web Intelligence and Agent Systems*, Jan 2008.

[8] J. Botia, J. Hernansaez, and A. Gomez-Skarmeta. On the application of clustering techniques to support debugging large-scale multi-agent systems. *Springer*, 2007.

[9] J. Botia, J. Hernansaez, and F. Skarmeta. Towards an approach for debugging mas through the analysis of acl messages. *MATES*, Jan 2004.

[10] J. Collis, D. Ndumu, H. Nwana, and L. Lee. The zeus agent building tool-kit. *BT Technology Journal*, Jan 1998.

[11] IEEE. *1003.1, 2004 EDITION IEEE Standard for Information Technology Portable Operating System Interface (POSIX)*. 2004.

[12] D. Lam and K. Barber. Debugging agent behavior in an implemented agent system. ... *Workshop on Programming Multi-Agent Systems at the Third* ..., Jan 2004.

[13] D. Lam and K. Barber. Comprehending agent software. *Proceedings of the fourth international joint conference on* ..., Jan 2005.

[14] D. Luckham. The power of events. *Addison-Wesley*, Jan 2002.

[15] V. Mafik and M. Pechoucek. Social knowledge in multi-agent systems. *Systems*, Jan 2004.

[16] D. Ndumu, H. Nwana, L. Lee, and J. Collis. Visualising and debugging distributed multi-agent systems. *Proceedings of the third annual conference on Autonomous* ..., Jan 1999.

[17] L. Padgham, M. Winikoff, and D. Poutakidis. Adding debugging support to the prometheus methodology. *Engineering Applications of Artificial Intelligence*, Jan 2005.

[18] A. Pokahr and L. Braubach. Jadex tool guide. page 66, Sep 2008.

[19] V. Sanchez-Anguix, A. Espinosa, L. Hernández, and A. García-Fornes. Mamsy: A management tool for multi-agent systems. *7th International Conference on Practical Applications of Agents and Multi-Agent Systems*, 2009.

[20] A. O. Software. Jack tm tracing manual. page 85, May 2008.

[21] P. Tichy and P. Slechta. Java sniffer 2.7 user manual. 2006.