Programming Multi-Agent Systems
A report of the technical forum meeting

Mehdi Dastani and Jorge J. Gomez-Sanz

Utrecht Universiteit, The Netherlands
mehdi@cs.uu.nl
Universidad Complutense de Madrid
jjgomez@sip.ucm.es
April 20, 2005

1 Introduction

The main motivation to organize the programming multi-agent systems technical forum (PROMAS TF) was to discuss and evaluate the problems and issues related to the implementation of multi-agent systems. In our view, agent oriented applications will only be taken up by industry if the gap between multi-agent system specification and design on the one hand and multi-agent system implementation on the other hand is bridged. In order to bridge this gap, at least some powerful and general purpose programming technologies should be developed such that the specifications and designs of multi-agent systems can be easily and directly implemented. This technology should include agent-based programming languages as well as tools that support multi-agent programming in a way that is usable and effectively applicable in industry.

The programming multi-agent systems technical forum aims at building bridges between industry and academy by creating an informal platform where they can present their works and discuss multi-agent systems implementation issues. In order to achieve this aim, we have organized the second meeting of the programming multi-agent systems technical forum in Ljubljana, Slovenia. As in previous editions, we had invited speakers from both industry and academy. We organized the meeting into seven sessions that dealt with both research and industry hot topics. The topics were formulated in terms of questions that made available to both speakers and attendants before the meeting. During the sessions the speakers and attendants had opportunity to discuss and answer the questions. Each session included one or two small presentations of about fifteen minutes followed by a dedicated discussion panel of about an hour where attendants found opinions of each other and could express their own. The topic
of the sessions and the speakers are summarized below.

- The role of Artificial Intelligence in multi-agent system developments
  - Ana Garcia Serrano, Universidad Politecnica de Madrid, Spain
  - Gabriel Marchalot, Thals group, France

- Development costs of MAS
  - Jorge Gomez-Sanz, Universidad Complutense de Madrid, Spain

- Imperative vs. declarative languages for multi-agent system implementation
  - Joao Leite, Universidade Nova de Lisboa, Portugal

- Formal methods in MAS
  - Rafael Bordini, University of Durham, U.K.

- Implementation of concrete multi-agent system issues
  - Birna van Riemsdijk, Utrecht Universiteit, The Netherlands
  - Allessandro Ricci, Universita di Bologna a Cesena, Italy

- IDEs for multi-agent system creation
  - Gregory O’Hare, University College Dublin, Ireland
  - Juan Manuel Serrano, Universidad Rey Juan Carlos, Spain
  - Lars Braubach, Universität Hamburg, Germany

- Programming principles in multi-agent system
  - Amal El Fallah-Seghrouchni, University of Paris 6, France

We were honored by a participative public, an average of thirty attendants who contributed significantly to a successful meeting. More details about the topics, questions, participants, and the program can be found on the following webpage of the PROMAS technical forum group:

http://www.cs.uu.nl/~mehdi/al3promas.html

In this report, we present for each session of the PROMAS meeting the questions that we had formulated before the meeting and made available to participants followed by a summary of the discussions that took place in each session. In the last section we present a brief conclusion of the meeting.

2 Artificial Intelligence and Multi-agent Systems

In the past it was hard to sell (academic) AI as such. However, the market is now ready for the (more mature) agent concept. How can both areas be integrated? What is the influence of AI concerns in a multi-agent systems? What is the experience in reusing AI libraries and software into multi-agent systems? Does AI really improve your multi-agent systems? Can a multi-agent system be built without AI? There should be a visionary road map (perhaps based on the
Agentlink road map), which indicates the maturity of the agent market and the need from academia. What can be learned from implementation and deployment of AI systems? Did AI need or result in special AI-oriented programming languages? Can AI-oriented programming languages be used for implementing multi-agent systems?

Since speakers in the last edition of PROMAS demoted the role of AI in agents, this edition wanted to balance the equation by inviting speakers who were researching precisely the role of AI in multi-agent systems. The speakers of this topic were Ana Garcia Serrano from the Universidad Politecnica de Madrid (Spain) and Gabriel Marchalot from Thals group (France). The former defended the presence of AI in multi-agent systems by means of knowledge engineering. Using Prolog as key language for AI, she presented two views about how knowledge engineering could participate in multi-agent system design. The first view dealt with knowledge to enable capacity for problem identification, internal capacity of problem solving, reasoning, and environment perception. The second view dealt with knowledge for modeling other agents and enact rational social decisions. To illustrate her point of view, she presented two applications implemented in her group: a shopping agent and a traffic control system. The second speaker presented AI techniques applied to an agent based monitoring system for maritime and airborne surveillance. He talked about concrete AI techniques that helped him in a real system. He discussed the problem of coordinating the perceptions of different sensors in order to understand what is being perceived exactly. Computationally, the problem was expensive, but by using different coordination mechanisms inspired in the work of multi-agent system coordination from Sascha Ossowski and classification trees, he managed to bring a satisfactory design and results.

This preliminary results were useful towards a discussion on AI and its role in multi-agent systems. Attendants focused on the problem of perceiving the environment and derived problems. One opinion was that perception ought to be embedded within agent decision procedure. One problem of perception is that it is hard to abstract symbols from perceptual data. This means investing more time before an agent can decide what to do. As an alternative, AI experience suggests to model only specific parts of the environment, those that, according to the agents’ design, are relevant for the designer’s objectives. This way, deriving symbols from perception becomes a more tractable problem.

The session finished with a quick question to the speakers: what is the relevance of AI in your research? Answers were equally positive and negative. Uses of AI-related languages, like Prolog, was made in a hybrid way by integrating Prolog based components with other imperative language based components. Here, Ana Garcia and Lars Braubach gave a positive answer. Amal El Fallah-Seghrouchni also had a positive opinion about AI, but not restricted to the symbolic approach. She remembered the work from Brooks and its relevance in works like self-organization. Other participants argued that agent community is not only an extension or part of the AI community, and that multi-agent
systems can be built without referring to AI techniques.

3 Development and Maintenance Costs of Multi-Agent Systems

Developing systems with agents should be less expensive, time and effort, than using other paradigms, or at least we intend it to be. How would you expect agent technology to decrease development costs? By increasing re-usability? By identifying concrete problems where agents, inherently, are better solution than other alternatives, say object oriented for instance? By producing more robust systems?

Most papers in the literature of agents claim that developing systems with agents is more better. There are arguments in favour of this claim. However, how much does it cost? Looking for answers, the speaker, Jorge Gomez-Sanz from Universidad Complutense de Madrid, commented his experience about implementation of multi-agents systems in three international projects. Multi-agent systems tend to be implemented as hybrid systems, where declarative and imperative languages get together. It is argued that the cost of developing multi-agent systems is usually higher than in a pure object oriented. Two reasons for the higher costs are the difficulties debugging the different parts of the system and the effort engineering two different paradigms. Surprisingly, this provoking statement did not cause any opposite reaction from the attendants. In fact, it became clear through different discussion panels that there was an agreement that most of the effort in multi-agent system development goes into the debugging part and that it is a very difficult and expensive task in the development of multi-agent systems.

Looking for more facts, it was explained that the development of multi-agent systems at the moment is not supported enough. There are Integrated Development Environments (IDE) for agents, but they do not provide effective debugging tools or proper links with specification tools. Maintenance costs, on the other side, may be inherently minor than with objects. Developers may decide to implement many agents with little functionality. Hence, adding new functionality would mean adding more agents without modifying existing ones.

4 Imperative and Declarative Programming Languages for Multi-Agent Systems

Despite the current proposals, imperative languages are chosen by most developers to produce their multi-agent systems. However, when talking about agent theories, researchers prefer to use declarative ones. Indeed, generic languages (not developed ad-hoc for multi-agent system development) such as Prolog, Oz,
Haskell, and Scheme could be used to develop such systems, but is it really more helpful? What agent developments do you know which are implemented with declarative languages? Pure declarative implementations or hybrid? How can declarative languages be used together with imperative ones? How can declarative languages facilitate the MAS development? It seems that it can be applied to define the high-level behavior of an agent. Where else?

The choice between declarative and imperative languages is one of the first decisions a developer has to take when facing the development of a multi-agent system. The speaker, Joao Leite, argued that the programming languages that are applied to implement multi-agent systems have been imperative languages. He emphasized the importance of declarative languages by making a quick review of existing ones and argued that declarative languages had not convinced the industry yet. Main problems towards acceptance were the theoretical purpose of these languages and their performance. Most people working with these languages are theoretical; hence they may not be interested in industrial applications. With respect to the performance and related with the previous problem, declarative languages for multi-agent system implementation are mainly prototypes. As a solution, Joao proposed to dedicate people to finding engineering applications of these languages and centralize effort. As an example, he mentioned the XSB effort, a Logic Programming and Deductive Database system for Unix and Windows.

The discussion panel raised questions about the role of declarative languages in multi-agent system implementation. Multi-agent systems are usually specified in terms of social and cognitive concepts such as interactions, roles, norms, groups, beliefs, goals, and reasoning about these concepts. Declarative languages can be useful in the implementation of these concepts and capabilities. Moreover, there were comments about the interest of promoting research on declarative goals. Experiences with students on using declarative languages was a topic that drew attention. Attendants who had expertise with 3APL commented that most students needed expressive data types for the implementation of multi-agent systems and that the Prolog part of the 3APL is usually considered as a limitation in this respect. It is also noted that students usually come from imperative language-oriented training, but when they first learn Prolog, they appreciate the integration of this language in 3APL. A problem is that people are not taught proper methods for programming Prolog.

Although declarative languages are essential for the implementation of certain aspects of multi-agent systems, imperative languages are indispensable for the implementation of other parts such as actions and plans. At the end, it was clear that programming languages for multi-agent systems need to be a mixture of different languages. To define various aspects of multi-agent systems, there must be a compromise between imperative and declarative languages.
5 Formal Methods for Multi-Agent Systems

There are significant amount of research in formal methods applied to agents. Among others, it could be mentioned model checking approaches, multi-agent system definitions based on temporal logic, situation calculus, and modal logic in general. Usually, formal methods are associated with high costs of developments since it requires time and high skilled workers. However, some kinds of applications require the application of these methods. Is easier to apply formal methods to multi-agent systems than to other paradigms? what advantages would you remark when applying formal methods to a multi-agent system development? can agents be a better way of introducing formal methods deeper into industry? or will agents be more easily uptake by industry due formal methods?

The application of formal methods to multi-agent systems has received much attention. In the last edition of this technical forum we discussed the application of formal methods to verify properties of protocols. The speaker of this session, Rafael Bordini from University of Durham, discussed the application of model checking to multi-agent systems. He introduced model checking techniques as the way actually followed by most researchers. According to him, there are several obstacles in solving the multi-agent system verification problem among which the determination of properties that need to be verified for multi-agent systems and the ability to work with medium size developments. It is a recent area for multi-agent systems which still needs to mature. One of the challenges is how to deal with the composition of single agent properties to proof properties of multi-agent systems. Another challenge about verification is dealing with agent autonomy. If we keep the autonomy of an agent, we cannot express a priori the behavior of the agent. After all, the agent is free to decide whatever is more appropriate. Taking into account the composition and the autonomy problems, it is still the question if an open multi-agent system can be verified at all. We can extend model checking to such systems. Probably, we will have to redefine the verification problem. Deadlock verification, for instance, is already known for communications, but can we still check deadlocks when autonomy is involved?

6 Multi-Agent Systems and Integrated Developing Environments

Possible links between multi-agent system integrated development environments (IDEs) and programming languages. Usually IDEs use a high level representation of the multi-agent system that are translated later into executable code. What representations are more useful? what translation mechanism are considered? In the executable code, which programming language is more appropriate?

Towards integration of multi-agent systems into industrial practices, it is important to have proper development environments that assist developer just
as we find tools that help us in using object oriented languages such as JAVA or C++. To have a picture of what is the current state in generic tools that help in producing multi-agent systems, three speakers were invited. First one was Gregory O’Hare from University College Dublin. He introduced his talk by remarking what is unique on multi-agent systems that demand new tools and methodologies. Many existing multi-agent systems do not address scalability, testing, and other deployment issues. Besides, if we intend to develop agents living in different computational nodes, Laptops and computers are not the only devices that can host agents; pda’s and other small mobile devices such as sensors can also host agents. They scale up and need to be implemented as multi-agent systems. Agent factory aims at giving support to all development phases and is trying to provide solutions in that direction.

Next speaker was Juan Manuel Serrano, from Universidad Rey Juan Carlos. His main question was how to generate JADE based codes based on a high-level specification of a multi-agent system. In particular, he discussed this issue by explaining how RICA meta-model (specification) can be used to generate RICA-J (JADE based) code. He explained the execution semantics of RICA model and discussed how this mapping takes place.

The final speaker of this session was Lars Braubach from Universität Hamburg. Lars discussed the IDE’s that have been proposed to develop object-oriented systems and explained their functions, their objectives and the lessons learned from those studies. He made then a quick review of the existing IDEs such as Jack, CAFnE, SOAR, AgentBuilder and AgentFactory and compared them in order to see advantages and disadvantages.

From the presentations and the panel discussion, it was commented that multi-agent systems are about integrating existing applications and legacy systems. Therefore, agent oriented IDEs still should evolve and need to cover many functionalities of object-oriented IDE’s. Moreover, the IDEs for multi-agent systems should support the specific characteristics of multi-agent systems such as coordination and environment which are not sufficiently studied yet. However, building IDE is difficult in general. For example, to produce IDEs that support the development of infrastructures for multi-agent systems is a hard challenge. Ideally, IDE should allow support and tools for developing different aspects of multi-agent systems.

7 Implementation of Multi-Agent Issues

There are many issues involved in multi-agent systems. At the multi-agent level, examples of such issues are coordination, communication, mobility, environment, security, ontology, and services. At the single agent level, examples are mental attitudes, agent roles, role dynamics, deliberation capabilities, adaptation capabilities, social capabilities, and reactive mechanisms. What are the techniques to implement these issues? Which languages provide programming constructs to implement them? How can these issues be implemented in exist-
ing programming languages such as Java or C++? In general, how can existing
MAS architectures, which are defined in terms of higher level concepts, be im-
plemented?

To have a spectrum of opinions about different features that are important
in the development of multi-agent systems, we invited two speakers. The first
speaker was Birna van Riemsdijk from Utrecht Universiteit who explained her
experiences related to the design of agent programming languages. She em-
phasized the importance of the semantics associated to the agent programming
languages and gave an overview of existing agent programming languages and
platforms. For each of these languages, she explained agent concepts that are
central in the design of the programming languages and discussed their seman-
tics. She noted that different notations and concepts are introduced at the
analysis and design phases of multi-agent system methodologies. This lead to
two important questions: which concepts have to be covered at the implementa-
tion phase and which programming constructs should be available to implement
them?

The second speaker in this session was Allessandro Ricci from Universita
di Bologna a Cesena. In his talk, he stated that interaction and coordination
is a source of complexity that we cannot avoid, even with coordination frame-
works. One of the major problems is how to debug a multi-agent system with
many agents interacting in a timely fashion. Debugging a coordination program
is a non trivial task. Besides, coordination in multi-agent systems is different
than coordination of processes (openness and heterogeneous). A way to help
defining and implementing coordination is by means of coordination artifacts
different from agent and environment. Hence, agents would use these artifacts
to coordinate their interactions. He argued that interaction is more general
than communication and that coordination and computation are different. To
give a better idea of what a coordination artifact is, he described the properties
of the coordination artifact, and proposed two levels of abstractions: agents
(goal/task) and artifact (services). To complement his presentation, he intro-
duced some of the existing coordination works on multi-agent systems.

The panel discussion drove its attention towards coordination problem. The
analysis and design phases of the existing multi-agent system methodologies
introduce organization based concepts. These concepts can be implemented in
terms of coordination mechanisms. A fundamental issue in developing multi-
agent systems is to separate the concern between agents and coordination ar-
tifact. In order to respect this separation of concern at the implementation
level, a programming language is needed that provide programming constructs
to implement coordination artifacts. Moreover, it is argued that the coordi-
nation and the environment are different entities and it should be decided at
the analysis and design phases where to model certain organizational aspects.
Finally, modifying coordination at the run time is very important and necessary
for multi-agent systems. The dynamic reconfiguration of coordination mechani-
isms is essential feature of many open multi-agent systems.
8 Programming Principles for Multi-Agent Systems

The development of multi-agent systems requires training in different conventional disciplines, such as object oriented programming, distributed systems, and structured programming. This has motivated the adaptation of some known techniques from those disciplines to multi-agent systems such as abstraction, inheritance, modularity, overloading, information hiding, error handling, generic programming, compositionality, and separation of concerns. However, it is hard to appreciate them nowadays since we find few available open source implementations of multi-agent systems and even less documentation. What public implementations of multi-agent system are known? What of the mentioned techniques that would be highlighted in those implementations? How to value the importance or impact of those techniques in a multi-agent system development? Do they influence more at Agent level or at multi-agent system level?

The maturity of a paradigm can be measured in the discovering of specific features that characterize it. Hence, object oriented paradigm can be considered as mature since most developers can mention that, for instance, they apply inheritance, polymorphism, and encapsulation. In order to have a multi-agent system paradigm, we need to locate similar features biased, of course, by what we think that characterize agents (e.g., autonomy, sociability, learning). In order to discuss these issues, we invited Amal El Fallah-Seghrouchni from University of Paris 6.

She characterized the space of possible existing multi-agent systems by three dimensions: AI, Interaction, and concurrency. Which techniques do we need for each of these aspects? She introduced some formal techniques that may help in this task. Petri nets are used successfully to represent and execute plans. Also, recursive Petri nets are used for protocols. Ambient algebra is used to implement individual agents. Process algebra is applied to agents by many researchers. Modularity in multi-agent systems is realized through individual agents as well as different modules within individual agents. Capabilities are the abstract view of modularity within agents.

The panel discussion was a general one where almost all topics that appeared along PROMAS were mentioned together. In developing multi-agent systems there is a need for dedicated software engineering methods. In this respect, some problems are, for example, how to translate concepts such as compositionality and inheritance to multi-agent systems. It is probably not desirable nor realistic to come up with a languages which integrate all techniques. Instead, different types of multi-agent programming languages are expected to emerge and applied. There are many concepts that are specific to multi-agent systems. These concepts should be fine tuned and their computational semantics should be defined and implemented. Multi-agent systems should become a software developing methodologies and therefore we need to find those abstractions.
9 Conclusions

AI appears as a way of improving agent behavior by making agents able to learn, plan or reason. On the other hand, AI also brings its classical problems, among which the problem of perception and generating internal representations of the world. In applying AI and trying to think about the implementation of multi-agent systems, declarative languages can be helpful, specially PROLOG, Lisp, and Haskell. Declarative languages are interesting because they are facilitating the implementation of mental state and reasoning capability of agents, but also the implementation of formal methods applied to multi-agent systems. However, current research is still at a theoretical level and the maturity of declarative languages still has not convinced most multi-agent system developers, who seem more devoted to imperative languages.

It was clear that formal techniques such as model checking are needed to test, debug and verify properties of implemented multi-agent systems. There were several opinions about which improvements should be first, remarking scalability, deployment, and debugging of multi-agent systems. Despite needed improvements, there seems to be a real need for defining the real difference in implementing systems using agent-oriented or object-oriented programming languages. It was argued that most multi-agent systems are implemented with object oriented languages; hence, where is the difference?

Speakers and participants emphasized the role of multi-agent system development environments which should help the development of complex multi-agent systems, new programming principles which should help to understand and to realize agent features, and formal semantics for multi-agent system programming languages which help to implement specific multi-agent system behavior. There was a general agreement that due to the lack of a multi-agent system dedicated programming languages and development tools building multi-agent systems are still a highly time and effort consuming activity. Finally, it is concluded that any multi-agent system programming language and development environment should support the development of three distinguished components involved in multi-agent systems: individual agents, their organization and coordination mechanism, and their shared environment.

This edition of PROMAS TF was rather productive. As a summary of the conclusions we could say:

- AI can play an important role in multi-agent systems, though its role still have to be refined
- At the moment, developing multi-agent systems involves higher costs than using conventional paradigms due to the lack of supporting methods and tools
- A balanced combination of declarative and imperative languages is desirable for implementing multi-agent system
- Formal methods need to be redefined in order to be properly adapted to multi-agent systems
• There are many IDEs that support the developments of multi-agent sys-
tems, but they are not suitable for large scale applications

• The programming languages for multi-agent systems should provide a va-
riety of programming constructs to implement specific high-level concepts
directly and effectively.

• When programming a multi-agent systems, applied solutions come from
three directions: AI, Interaction, and concurrency.

Many interesting discussions, aside from the presented ones, appeared during
the event. We tried to gather as many discussed issues into this report. Unfor-
tunately, it was not possible to cover the entire discussion in detail.