The Knowledge Engineering Review, Vol. 20:2, 151–164. © 2005, Cambridge University Press
doi:10.1017/S0269888905000482 Printed in the United Kingdom

Programming multi-agent systems

MEHDI DASTANI[1] and JORGE J. GOMEZ-SANZ[2]

[1]*Utrecht Universiteit, The Netherlands;*
E-mail: mehdi@cs.uu.nl
[2]*Universidad Complutense de Madrid, Spain;*
E-mail: jjgomez@sip.ucm.es

Abstract

PROMAS[1] (Programming Multi-Agent Systems) is an AgentLink[2] technical forum that aims to bring together the researchers and practitioners from both academia and industry to discuss the problems related to the development of multi-agent systems and to evaluate the existing proposals and results. The specific focus of this technical forum is on agent-oriented programming languages and tools that can effectively and efficiently support the implementation of multi-agent systems. This paper provides an overview of the main contributions and conclusions derived from the first two editions of PROMAS technical forum.

1 Introduction

Agent-oriented software engineering is a new paradigm in the field of software engineering (Wooldridge & Jennings, 1995; Jennings, 2000). It aims at developing and engineering multi-agent systems for complex distributed applications through powerful and natural high-level abstractions. Typical examples of complex distributed applications are social simulations, supply-chain management, electronic auctions, e-institutions, or business-to-business applications. Multi-agent systems can be used by developers to naturally model and develop software for such applications.

From the software engineering perspective the systematic development of software should follow several phases. The number and purpose of the phases depend on the problem domain and the methodology. In general, it is easy to find requirement elicitation, system analysis, architectural design, implementation, and testing phases in most software development methodologies. There have been many agent-oriented methodologies proposed, like GAIA (Zambonelli *et al*., 2003), Prometheus (Padgham & Winikoff, 2003), Tropos (Bresciani *et al*., 2003), SODA (Omicini, 2000), MAS-CommonKADS (Iglesias *et al*., 1998), or INGENIAS (Gomez-Sanz & Pavon, 2003). Each of these methodologies focuses on the role of agent abstractions on certain phases of the multi-agent system development. However, the main focus of most of these agent-oriented methodologies has been on the analysis and design phases, paying less attention to the implementation of multi-agent systems.

Multi-agent systems will only be taken up by industry if the gap between multi-agent system analysis/design and multi-agent system implementation is bridged. To achieve this goal, some powerful and general purpose programming technologies should be developed such that the specifications and designs of multi-agent systems can be easily and effectively implemented. These technologies would include agent-based programming languages as well as tools that support multi-agent programming.

[1] The Webpage for PROMAS technical forum is `http://www.cs.uu.nl/~mehdi/al3promas.html`.
[2] The Webpage for AgentLink is `http://www.agentlink.org/`.

https://doi.org/10.1017/S0269888905000482 Published online by Cambridge University Press

Fortunately, in the past several years we have witnessed an increasing maturity in the implementation of multi-agent systems. This maturity is due to the development of robust programming techniques, principles, and tools. Among them, we could mention the importance of agent-oriented programming languages, coordination techniques, reasoning engines, agent architectures, multi-agent frameworks, and integrated development environments for multi-agent systems. This progress has been achieved, in part, due to the growing number of events on issues related to the implementation of agent-oriented systems (Dastani *et al*., 2003; Bordini *et al*., 2005a). These elements are contributing strongly to agent technology approaching its industry strength applications.

The main motivation to organize the programming multi-agent systems technical forum, PROMAS TF from now on, is to bring together researchers and practitioners from academy and industry to discuss the problems related to the implementation of multi-agent systems and to evaluate the existing proposals and results. This forum contributes to the collaboration between industry and academy by creating an informal platform where they can:

- identify what is needed for developing programming languages and tools that can effectively support multi-agent programming;
- explain how the multi-agent technology can be presented to industry and effectively applied in industry practices.

Though simple in appearance, these two tasks imply reviewing a huge amount of research works and discussing many open problems and research directions. In our view, designing agent-oriented programming languages, developing implementation tools, sharing them, discussing them, and testing their viability in industry practices is a must at this moment. PROMAS TF did initiate several tracks in different research directions and succeeded in organizing two editions[3] with participants from industry and academy. These two editions of PROMAS TF have established links between industry and academy.

In this paper, we present the results and discussions of the first two editions of this technical forum. They have been structured into three sections. Section 2 deals with efforts in understanding the concepts underlying any multi-agent system implementation and reviews three major influences on multi-agent systems: artificial intelligence, object-oriented software engineering, and coordination/concurrency research. Section 3 provides an overview on recent research in producing agent-oriented programming languages and development tools that make it easier to develop multi-agent systems. Finally, Section 4 explains the steps taken in order to convince industry of the benefits of agent technology. We conclude the paper with some remarks about the topics discussed in the first two editions of PROMAS TF and comment on the future work.

2 Multi-agent systems: a multidisciplinary field

One of the concerns of the first PROMAS TF meeting was the role of other computer science disciplines such as artificial intelligence, object-oriented software engineering, and coordination/concurrency in multi-agent system developments. In this section, we give a brief summary of the discussions about these topics.

2.1 Artificial intelligence and multi-agent systems

Formal methods and techniques from artificial intelligence have influenced the design of agent-oriented programming languages and tools. One of the early concerns of artificial intelligence was the problem of knowledge representation and reasoning. Many logical and probabilistic formalisms have been proposed to represent and reason about knowledge. These formalisms have been used in

[3] The third edition of PROMAS TF will take place in Budapest on 15 September 2005

https://doi.org/10.1017/S0269888905000482 Published online by Cambridge University Press

multi-agent systems to specify and implement different aspects of autonomous agents, such as their beliefs and goals. For example, in the agent-oriented programming languages 3APL (Dastani *et al.*, 2004b) and Jason (Bordini *et al.*, 2005c) the beliefs and goals of individual agents are represented by first-order (atomic) formulae. Various resolution techniques and unification methods are used to allow reasoning about beliefs and goals. In other agent programming languages such as Jadex (Pokahr *et al.*, 2005), the beliefs of agents can be represented either by the usual data structures such as arrays and objects or by tables or databases. In the latter case, the beliefs of an agent are examined by means of SQL-like queries. Moreover, the beliefs and goals of individual agents can change due to their observations and interactions. Different update, revision, and merging techniques can be used to manage the dynamics of beliefs and goals. For example, in 3APL and Jason simple mechanisms such as addition and deletion of first-order atomic formulae have been used to update an agent's beliefs and goals.

Planning is yet another technique from artificial intelligence which plays an important role in the design and implementation of individual agents. Most agent programming languages such as 3APL, Jason, Jadex and Jack (Bordini *et al.*, 2005b) are designed to implement reactive agents. These languages allow a programmer to implement the connection between goals and plans, i.e. a programmer can indicate which plan should be adopted and executed to achieve certain goals. However, there are no dedicated programming constructs to implement a process that generates plans based on reasoning about available actions and the current context. On the other hand, other agent-oriented programming languages, such as FLUX (Thielscher, 2005) and ConGolog (De Giacomo *et al.*, 2000) are designed to tackle the planning problem by providing specific programming constructs to implement agents that can reason about their actions and plans.

Another result from artificial intelligence that has been adopted for the development and implementation of multi-agent systems is knowledge engineering. According to attendees from the Technical University of Madrid, the role of knowledge engineering can be studied from two points of view. The first view deals with knowledge to enable capacity for problem identification, problem solving, reasoning, and environment perception. The second view deals with knowledge that can be used to model other agents and based on which rational social decisions can be taken. Based on this view on knowledge engineering, some applications have been developed by the Technical University of Madrid. These applications cover different domains like shopping agents (García-Serrano *et al.*, 2001) or traffic control systems (Ossowski *et al.*, 2004).

Other results from artificial intelligence applied to implement multi-agent systems are probabilistic classification trees, distributed decision techniques, and negotiation mechanisms. The representative from the French Thales group (`http://www.thalesgroup.com`) explained how these techniques were used to implement an agent-based monitoring system for maritime and airborne surveillance. In this application, each sensor is modelled as one agent. One problem in this application is how to coordinate the inputs of different sensors in order to understand what is being perceived. Different coordination mechanisms, distributed decision techniques, probabilistic classification, and decision trees are used to manage the complexity of this problem. Despite these approaches, it is observed that social metaphors and agent-based concepts are very useful to understand and develop cooperation and coordination between sensors.

Related to the representation problem, one of the questions that emerged during the PROMAS meetings was how agents in industrial applications should/can perceive their physical/external environment, since it is hard to abstract symbols from perceptual data. This abstraction process implies investing more time to understand the environment before an agent can decide what to do. The experience of the PROMAS attendants suggested to model and perceive only relevant parts of the environment, i.e. those parts that are relevant for the objectives of the designers of the agents. In this way, deriving symbols from perceptual data becomes a tractable problem. Another question related to the perception of an agent is whether the agent needs to perceive the environment continuously or not. In many agent architectures, such as the PRS-inspired architectures (Ingrand *et al.*, 1992), an agent will perceive the environment only once in its deliberation cycle. In other

https://doi.org/10.1017/S0269888905000482 Published online by Cambridge University Press

agent architectures, such as the subsumption architecture (Brooks, 1991), an agent can continuously perceive the environment. Deciding which agent architecture to use is thought to depend on the problem domain.

2.2 *Object-oriented software engineering and multi-agent systems*

Many agent-based systems deployed in industry are implemented directly in object-oriented languages, especially Java. Well-known agent programming languages and tools, such as Jade, Jack, and Jadex, are extensions of Java, or implemented in Java, e.g., Jason and 3APL. Taking as a reference the list of agent-related software published in AgentLink (`http://www.agentlink.org/`), it can be observed that the number of systems implemented with an object-oriented language is greater than the number of systems implemented with a non-object-oriented programming package.

The reason for using object-oriented programming languages to implement multi-agent systems is thought to be the close similarity between objects and agents. This similarity makes it easier to implement agents with objects. In fact, some researchers in the field of multi-agent systems believe that the concept of agent is only an abstraction and that there is no special need for introducing a new agent-oriented programming languages. They argue that multi-agent systems can be implemented directly in object-oriented languages as long as, for example, interaction is implemented by asynchronous communication instead of method calls.

There are indeed many factors that contribute to the similarity between agents and objects. First, objects are adequate for the design of complex architectures. Due to the inherent encapsulation and polymorphism of objects, the combination of different objects as building units is easier. These object features can be exploited to detail agent architectures and to implement complex applications. Second, object-oriented programming languages such as Java support program mobility by means of serialization of codes. This feature is an effective means for the implementation of mobile agents. Third, most middleware for distributed systems is built with object-oriented languages. Although non-object-oriented programming languages have also developed constructs to deal with the distribution of code, it cannot be denied that object-oriented programming languages form the first alternative when building a distributed system. Current middleware enabling code distribution, such as RMI, CORBA, .NET, J2EE, are examples of object-oriented solutions. Therefore, since multi-agent systems tend to be distributed, it is natural to implement them in an object-oriented programming language. Fourth, objects are entities with state. The concept of agent can be considered as an extension of the concept of object by adding extra features such as the control of its own state. In the case of BDI agents, the state of an object can be specified as the mental state that consists of mental attitudes such as beliefs, desires, and plans. Finally, one of the strongest arguments in favour of using object-oriented programming languages to implement multi-agent systems is the availability of sophisticated software development tools. Most well-known object-oriented languages have had robust and flexible integrated development environments (IDEs) for a long time. For more discussion about IDEs, refer to Section 3.4.

After this propaganda supporting the implementation of multi-agent systems in object-oriented programming languages, it could be argued why we need to design agent-oriented programming languages. Agents require computational representations for, among other things, autonomy, social and cognitive behaviour, and mobility. Implementing these features in objects requires that the software developer has the interdisciplinary knowledge and techniques we mentioned at the beginning of this section. Agent-oriented programming languages, despite being implemented with objects, provide specific programming constructs to implement agent-specific features. The challenge today is finding the proper programming constructs for agent-specific features. In any case, confusing agents and objects is something to be superseded by new generations of developers. The more developers work with them, the easier it will be to comprehend the differences between objects and agents, not only in theory but also in practice.

https://doi.org/10.1017/S0269888905000482 Published online by Cambridge University Press

2.3 Concurrency, coordination and multi-agent systems

Complex distributed applications can be understood and analysed as multi-agent systems consisting of autonomous agents whose interactions are coordinated through an organizational structure. The organizational structures are usually described and analysed in terms of social concepts and relations such as roles, permissions, obligations, norm, trust, power, delegation of tasks, responsibilities, permissions, access to resources, and communication (Ferber & Gutknecht, 1998; Ferber *et al.*, 2003; Kolp *et al.*, 2002; Castelfranchi, 1990; Wooldridge & Jennings, 1995; Dastani *et al.*, 2004a; Grossi *et al.*, 2005). Little attention has been paid to the *implementation* of social and organizational aspects of multi-agent systems. In order to design programming languages to implement such organizational concerns, computational theories and semantics for social/organizational concepts and relations are indispensable. In this line, seminal works such as Sichman *et al.* (1994) and Baldoni *et al.* (2005) can be consulted.

The theory of concurrency and coordination is relevant for the development formal and computational semantics of social and organizational concepts. These concepts can in turn be used to design programming languages that facilitate the implementation of organizational structures of multi-agent systems. A rich variety of possible extensions and generalizations of existing exception-handling and event-based mechanisms can be used for modelling various basic interaction patterns that implement social concepts and relations. For example, it is possible to define an interaction pattern between two agents A and B by specifying certain actions of A that will be triggered by errors generated by certain actions of B for which A is responsible.

Coordination languages and infrastructures such as Linda (Gelernter, 1985), Manifold (Arbab *et al.*, 1993), Reo (Arbab, 2004), and TuCSoN (Ricci *et al.*, 2001), can be used (modified and extended if necessary) in order to define programming constructs for the implementation of social concepts and relations. The advantage of adopting existing coordination languages and infrastructures is that they have computational semantics. Therefore, the computational semantics of organization structures and social concepts and relations can be specified in terms of computational semantics of coordination languages. Moreover, an important issue in the coordination studies is the dynamics of coordination. Using coordination mechanisms to implement organizational structures, the dynamics of coordination can model dynamic reconfiguration of organizations, i.e. how organizational structures progress. These dynamics can be defined on the basis of event-generation and event-handling mechanisms.

3 Programming languages and development tools for multi-agent systems

A fundamental principle in developing multi-agent systems is to separate different concerns: individual agents, the organizational structure that coordinates their activities, and their external environment. In order to respect this separation of concerns at the implementation level, agent-oriented programming languages should provide specialized programming constructs to implement different issues within these three aspects separately. The issues related to individual agents are beliefs, goals, plans, and a decision-making process that deliberates on these entities to determine the actions to be performed. At the environment level, the issues are the effects of the actions that are performed by the agents and the access to resources and services. Finally, at the multi-agent and social level, the issues are coordination of actions, communication, mobility, and security. These issues can be implemented by means of coordination artifacts, which are intended to be used by individual agents to coordinate their activities. The coordination artifacts facilitate interaction services and allow individual agents to access resources and services.

The existing programming languages and development tools for multi-agent systems can be distinguished by means of programming constructs and functionalities that they provide to facilitate the implementation of the mentioned issues. It should be clear that multi-agent systems can be implemented with existing programming languages and development tools such as Java, Prolog, C++, but the point is that agent-oriented programming languages and tools provide

https://doi.org/10.1017/S0269888905000482 Published online by Cambridge University Press

dedicated programming constructs and functionalities that facilitate the direct and effective implementation of the aforementioned issues.

The agent-dedicated programming constructs and functionalities are defined by means of formal methods and techniques such as operational semantics, reasoning techniques, update operations, and model checking. In particular, it is crucial that the provided concepts in agent-oriented programming languages have exact and computational semantics. A technique used, for example in 3APL and Jason, to define the exact and computational semantics of these concepts is operational semantics (Plotkin, 1981). It is also important to debug implemented multi-agent systems and to check if the implemented systems satisfy certain properties. Although model checking and theorem proving are the well-known techniques for debugging purposes, model checking is often used to debug multi-agent systems (Wooldridge *et al.*, 2002; Penczek & Lomuscio, 2003). One of the major problems is how to debug a multi-agent system with many agents interacting in a timely fashion. Various types of monitoring tools, such as sniffers and observation windows, have been proposed to follow the execution of multi-agent systems and to observe their behaviour.

3.1 Programming principles for multi-agent systems

Can we formulate some (theoretical and practical) requirements that programming languages, tools, and platforms for multi-agent systems should satisfy in order to make them effective for academy and industry practices? In other words, which programming principles should be supported by agent-oriented programming languages, tools, and frameworks? In general, the maturity of a software paradigm can be measured by means of its specific characterizing features that can be applied to specify, design, and implement a system. Hence, object-oriented paradigm can be considered as mature since features such as inheritance, polymorphism, and encapsulation can be applied at various phases of the object-oriented software development process. In order to have an agent-oriented system development paradigm, we need to identify specific agent-oriented characterizing features, such as autonomy, sociability, learning, heterogeneity, reusability and encapsulation, and indicate how they can be applied to develop multi-agent systems.

The development of multi-agent systems requires training in different software engineering paradigms, such as object-oriented programming, distributed and concurrent programming, constraint programming, and structured programming. This has motivated the adaptation of some well-known programming techniques and principles from those paradigms to multi-agent systems. Examples of such techniques and principles are abstraction, inheritance, modularity, overloading, information hiding, error handling, generic programming, compositionality, and separation of concerns. For example, modularity in multi-agent systems is supported through individual agents as well as different modules within individual agents such as capability and role in Jadex and 3APL. Also, import and export of capabilities support inheritance in Jadex, and the composition of coordination artefacts supports compositionality in TuCSoN and Reo. It is probably neither desirable nor realistic to come up with agent-oriented programming languages that integrate all techniques and support all programming principles. Instead, different types of agent-oriented programming languages are assumed to emerge.

3.2 Imperative and declarative programming languages for multi-agent systems

The existing agent-oriented programming languages are inspired and implemented by different programming paradigms such as imperative, declarative, or a combination. In general, an imperative program consists of a sequence of commands that describe how to solve a problem while a declarative program consists of a set of statements that describe what is the problem. Each programming paradigm has certain advantages and disadvantages. For example, imperative programs can have fast execution while declarative languages are more general, shorter, and readable.

https://doi.org/10.1017/S0269888905000482 Published online by Cambridge University Press

The choice between declarative and imperative languages is one of the first decisions a developer has to take when facing the implementation of a multi-agent system. As argued by invited speakers during the second edition of PROMAS TF, the programming languages that are applied to implement most sophisticated multi-agent systems have been imperative (object-oriented) languages (see also Section 2.2). The importance of declarative languages is emphasized by explaining that many agent-oriented concepts and issues can best be implemented in a declarative way. Agent-oriented programming languages that are inspired by declarative programming languages are reviewed and discussed. Examples of these programming languages are ConGolog, MetateM (Fisher, 1993), DyLog (Baldoni *et al.*, 2000), Flux (Thielscher, 2005), DALI (Costantini & Tocchio, 2002), MINERVA (Leite *et al.*, 2002), ALIAS (Ciampolini *et al.*, 2003), and AGENT-0 (Shoham, 1993). The first seven agent-oriented programming languages are implemented in Prolog while the last one is implemented in Common Lisp. Other agent-oriented programming languages such as IMPACT, Jack, Jason, and Jadex are implemented in Java, which is an imperative programming language. Finally, programming languages such as 3APL and PROSOCS (Stathis *et al.*, 2004) are implemented in a combination of imperative, (i.e. Java) and declarative, (i.e. Prolog) programming languages.

It is argued that declarative programming languages have not convinced industry yet. The main problems towards the acceptation of declarative languages are thought to be the theoretical purpose of these languages and their performances. Most people working with these languages have theoretical backgrounds and may not be interested in industrial applications. However, declarative languages are often used to implement multi-agent system prototypes. It is proposed to centralize effort and dedicate people to find engineering applications of these languages. This was made in the XSB initiative, a Logic Programming and Deductive Database system for Unix and Windows (`http://xsb.sourceforge.net`).

On the other hand, multi-agent systems are usually specified in terms of social and cognitive concepts such as interactions, roles, norms, groups, beliefs, goals, and reasoning. Declarative languages can be useful in the implementation of these concepts and capabilities. For example, in many programming languages such as 3APL and Jason the beliefs and goals of agents are implemented declaratively as a set of Prolog-like statements. The corresponding reasoning engines make it possible to reason about beliefs and goals.

Experiences with students using declarative languages formed a topic that drew attention. One of the lessons from these experiences is that declarative languages do not support expressive data types (such as array and object) and that this limitation is a source of problems for the implementation of sophisticated multi-agent systems. Another experience is that students usually come from imperative language training and they are not taught proper methods for declarative programming. Finally, it should be noted that students who are familiar with declarative programming languages appreciate hybrid agent-oriented programming languages, i.e. agent-oriented programming languages that are inspired by an integration of declarative and imperative programming paradigms. Although declarative languages are essential for the implementation of certain aspects of multi-agent systems, such as the beliefs and goals of individual agents, imperative languages are indispensable for the implementation of other aspects of agents such as actions and plans.

3.3 Formal methods for multi-agent systems

By formal methods, we mean here mathematically based techniques for the specification, implementation and verification of software. There is a significant amount of research in formal methods applied to agents, among others, model checking approaches, multi-agent system specification based on temporal logic, situation calculus, Petri-nets, and modal logic in general. Being an agent implementation technical forum, there was interest in the contribution of formal methods to the implementation and verification of multi-agent systems. Usually formal methods are

https://doi.org/10.1017/S0269888905000482 Published online by Cambridge University Press

associated with high development costs since they require time and highly skilled workers. Nonetheless, there are critical applications, such as control systems for power plants, that require such formal methods to ensure robustness and security. An example of the application of formal methods to multi-agent systems is the verification of interaction protocols. There are several obstacles in extending this verification to complete multi-agent systems, like determining which properties need to be verified and the ability to work with medium sized developments. Another challenge in verifying multi-agent systems is how to deal with the composition of properties of individual agents to proof properties of multi-agent systems. There is some work in this line, e.g., Jonker & Treur (1998).

Finally, verification of multi-agent systems should deal with agent autonomy as well. For example, can we specify and verify *a priori* the behaviour of an *autonomous* agent? After all, the intuition behind autonomous agents is that they are capable of deciding what to do based on their internal states and their observations. The verification of an autonomous agent may thus require the simulation of its environment, including other autonomous agents and the external/physical environment. These tasks are, however, not trivial. Taking into account the composition and the autonomy problems, there is still the question of what kinds of properties of multi-agent systems can be verified. Research on coordination artefacts and infrastructures may be helpful as they implement and guarantee the system level behaviour regardless of the autonomous behaviour of individual agents. Model checking, a properties verification technique, can be extended to verify properties of multi-agent systems, but most probably, it will require redefining the verification problem in terms of properties of individual agents, properties of coordination artefacts, and properties of their interactions.

3.4 Integrated developing environments for multi-agent systems

Towards a successful integration of multi-agent systems into industrial practices, it is important to have proper integrated development environments (IDEs) that assist software developers to build multi-agent systems just as we find development environments that help us in using object-oriented languages such as JAVA or C++ to implement software applications. Examples of integrated development environments that are proposed for multi-agent systems are Jack (Winikoff, 2005), CAFnE (Jayatilleke *et al*., 2005), SOAR (Laird *et al*., 1987), AgentBuilder and AgentFactory (Ross *et al*., 2005).

The question is which functionalities should be integrated in agent-oriented development environments. An example of such functions is the translation of high-level specifications of multi-agent systems into executable codes. In order to add such a functionality to an agent-oriented development environment, tools such as RICA-J JADE (Serrano & Ossowski, 2004) can be employed. This tool can generate JADE code based on a high-level specification of multi-agent systems. Moreover, it is desirable that agent-oriented development environments can support scalability and heterogeneity aspects of multi-agent systems. For example, if we intend to develop agents living in different computational nodes, laptops and computers are not the only devices that can host agents. PDAs and other small mobile devices, such as sensors, can act as hosts as well. Such systems can scale up to large multi-agent systems with different types of agents and based on different (physical) platforms. Agent factory is an example of an agent-oriented IDE that aims at giving support to the development of such large-scale heterogeneous multi-agent systems.

It is argued that multi-agent systems are required to coexist with other applications and legacy systems. This suggests that a development environment for multi-agent systems should have functionality to support the development of legacy systems as well. Currently, agent-oriented development environments should still evolve to cover many functionalities that are present in object-oriented development environments. Moreover, the integrated development environments for multi-agent systems should support the development of various ingredients in multi-agent systems, such as individual agents, coordination artefacts, environment, and services. Some of these

https://doi.org/10.1017/S0269888905000482 Published online by Cambridge University Press

ingredients are not sufficiently studied yet. Finally, there is a lack of experience about the kind of graphic user interfaces that are needed to interact with developers. In general, it can be concluded that building an integrated development environment for multi-agent systems is much harder than for object-oriented systems.

4 Convince industry to adopt multi-agent system technology

Most papers in the multi-agent systems literature claim that developing systems with agents is better. It is argued that agents are better because they are autonomous, robust, or more intuitive to understand. These arguments are necessary but not sufficient for convincing industry to adopt multi-agent technology. In this section, we give an overview of some factors that are important for industry to adopt multi-agent technology. Readers interested in further factors and evidence should consult Luck *et al.* (2003).

4.1 Awakening industry expectations

There is a growing interest in applying multi-agent technology in industry. For example, multi-agent systems are developed for insurance companies, call centres, and telecommunication companies. Responding to the question of what are typical multi-agent applications in industry and how do we convince industry to adopt multi-agent systems, the representative of Acklin B.V. wrote the following:

> A lot of processes within the Banking and Insurance domain can be made more efficient with the use of agent technology. The company Acklin had developed a number of agent-based systems that streamline steps in the claim handling process. In order to convince companies to use systems on the basis of agents, a number of 'commercial issues' need to be dealt with. These issues include: 'Why should we use something new?', 'Are agents proven technology?', 'Is there a strong business case, i.e., what is the problem? what is the return of investment'.

It is important to notice that some multi-agent applications are developed for the end users, while others are used off-line for simulation or prototyping purposes.

According to the representatives from industry, they need innovative techniques and approaches, such as multi-agent technology, as long as there are means to manage their complexities and they provide benefits in the short or long term. In order to apply multi-agent technology, industry requires computationally tractable theories for many multi-agent issues, such as planning, decision-making, or reasoning about high-level concepts such as knowledge and goals. It is observed that each company that adopts multi-agent technology has its own *ad hoc* implementation of the usual agent architectures. Examples of these companies are Calico Jack, Agentis, and Acklin. There are many reasons why agent companies develop their own technology, e.g., the demands and constraints of customers (working with existing customer systems and software developers, and the policies for software development), the fact that software developers need to be familiar with the software and trust it, and the need for dedicated solutions. It is unclear whether this is a positive and natural thing or just a consequence of the fact that there are no standard or academic tools with the right theoretical and practical basis.

4.2 Development and maintenance costs

Can the software development cost be reduced by adopting multi-agent systems instead of a different technology, for example, object-oriented technology? This is a simple question that in conventional software engineering, like object-oriented systems, cannot be answered without difficulties. For the industry, it is important that developing systems with agents is less expensive (in

https://doi.org/10.1017/S0269888905000482 Published online by Cambridge University Press

time and effort) than with other paradigms. The current experiences with industry show that they are not willing to pay for technology, but for solutions to specific problems. Moreover, they need to see rapid and substantial return on investment. The central question is how agent technology can decrease development and maintenance costs? Some standard cost reduction factors are software re-usability, identification of concrete problems where agents are inherently better solutions than other alternatives, and producing more robust systems.

It is argued that the development cost of multi-agent systems is currently higher than the development cost of pure object-oriented systems. There are several reasons supporting this statement. One is the difficulty with debugging different parts of multi-agent systems. It became clear through PROMAS TF editions that most of the effort in multi-agent system development goes into the debugging part. Another reason for the higher costs is the inadequacy of multi-agent system supporting tools compared with those for object-oriented technology, as is mentioned in Section 2.2. Finally, multi-agent systems tend to be implemented as hybrid systems, where declarative and imperative approaches get together (see also Section 3.2). The effort required in engineering these two different paradigms seems to be yet another reason for the higher development cost. On the other hand, the maintenance costs of multi-agent systems may be inherently less than the maintenance costs of object-oriented systems. Developers may decide to implement many agents, each one having had a piece of the whole functionality. Hence, adding new functionality would mean adding more agents without modifying existing ones. Also, there is some evidence, based on software engineering cost estimation models, that avail lower costs of agent-oriented developments versus object-oriented ones, (Gomez-Sanz *et al*., 2005). Unfortunately, the lack of feedback on industrial agent applications does not help in confirming this or any other assumption.

4.3 Applying multi-agent systems depends on company

The adoption of multi-agent technology by industry depends very much on the type of companies. According to the representative of Calico Jack Ltd:

> It is vital to distinguish between on the one hand, agent-based small and medium-sized enterprises (SMEs) and R&D groups of large multi-nationals, and on the other hand, the operational units of those multi-nationals, and other large companies. The latter group are the customers for agent technology: they are paying for solutions to problems, and need to show rapid and substantial return on investment. The former group are not the customers: they are struggling just as hard to sell multi-agent systems as are the academic community. So there is a clear route to alliance between academics, SMEs, and R&D groups. This alliance needs to develop a coherent and persuasive story with which to sell to industry proper—only then can the development of multi-agent systems in the real world really take off in the way that we are all striving for.

The industry can thus be classified into two categories: the companies that produce multi-agent solutions and those that consume multi-agent solutions. The customer companies, such as insurance companies or call centres, need software for their applications. They usually do not care whether the developed software is based on multi-agent technology or not. The producer companies, such as Acklin or Calico Jack, develop software for their customers. They do care about the methodology since it affects the efficiency of their software development process and their revenue. They usually do not tell their customers which technology they use unless their customers have explicit constraints or demands on the developments and the use of the software. For example, in some cases the developed software for some customers will be embedded in (and integrated with) existing systems. In such cases, the customer requirements determine which technology should be used. The experience of PROMAS TF industry representatives shows that the managers of the customer companies are usually not aware of the multi-agent system technology and that they are

https://doi.org/10.1017/S0269888905000482 Published online by Cambridge University Press

very careful/hesitant of adopting a new technology. In contrast, the people from the IT departments of the customer companies are interested in understanding and using the multi-agent system technology.

4.4 Teaching and transferring multi-agents systems to industry

Academia can help in promoting multi-agent systems and solving the problems that hinder industry to adopt multi-agent systems. Some of these problems need technological solutions, but other problems can be solved through education. Multi-agent technology, its applications, and advantages should be taught to industry. Moreover, the academic community can play an important role in multi-agent technology transfer by providing computationally tractable theories and tools. For example, industry needs tools that can be used to analyse applications, formulate requirements, and specify, design, implement, test and deploy multi-agent systems.

In this respect, the representatives of Calico Jack Ltd have noticed:

> As a small company selling agent-based solutions, Calico Jack Ltd has had to formulate its own response, and there may be general lessons that can be learnt. The first is that lumping agents research in with AI may be unproductive, especially when talking to industry: AI is a label by which to classify and pigeon-hole—and it is not a classification that is well funded. The second is that in the current industrial climate, CEOs are not paying for technology. Instead, they are paying for specific solutions. By its very nature, multi-agent systems represent a horizontal technology, so tailoring is crucial: it cannot just be 'agents', it must be 'an agent-based solution for doing X', where at its most general, X is a narrow market segment.

5 Conclusions

Results from artificial intelligence improve agent behaviour by making agents able to learn, plan or reason. On the other hand, artificial intelligence also brings its classical problems, among which the problem of perception and how to generate internal representations of the world based on the observed input data. In applying artificial intelligence and trying to think about the implementation of multi-agent systems, declarative languages can be helpful, especially PROLOG, Lisp, and Haskell. Declarative languages are interesting because they facilitate 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.

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, including scalability, deployment, and debugging of multi-agent systems. Despite needed improvements, there seems to be a real need for defining the 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 of the last two editions of PROMAS TF emphasized the role of multi-agent system development environments, which should help in the development of complex multi-agent systems. New programming principles should help us to understand and realize agent features. Formal semantics for multi-agent programming languages help to implement exact multi-agent system behaviour. There was a general agreement in the meetings that due to the lack of dedicated multi-agent programming languages and development tools, building multi-agent systems are still a highly time and effort consuming activity. It is concluded that multi-agent system programming languages and development environments should support the development of three distinguished components involved in multi-agent systems: individual agents, their organization and coordination mechanism, and their external environment. Finally, it is emphasized that a better

https://doi.org/10.1017/S0269888905000482 Published online by Cambridge University Press

understanding of the needs of industry is crucial for the development of multi-agent technology. The industry should clarify their needs by providing case studies, application types, and requirements. Some of the participants from industry indicated that they have case studies and applications for which they have already provided multi-agent technology solutions. They are willing to communicate these to other participants (van Aart *et al*., 2002, 2004).

PROMAS TF editions were rather productive. The following could be given as a summary of the conclusions reached.

- Artificial intelligence can play an important role in multi-agent systems, though its role still has to be refined.
- Most practical agent applications are in the area of system integration and require little or no intelligence as yet.
- The programming languages for multi-agent systems should provide a variety of programming constructs to implement specific high-level concepts directly and effectively.
- A balanced combination of declarative and imperative languages is desirable for implementing multi-agent systems.
- There are many integrated development environments that support the development of multi-agent systems. However, their capabilities to deal with large-scale applications still have to be demonstrated.
- Agent development tools should allow an easy integration of an agent with other applications. Only after this is accomplished, more intelligence will be needed.
- Model checking methods need to be redefined in order to be properly adapted to multi-agent systems and their specific features, such as autonomy.
- At the moment, developing multi-agent systems involves higher costs than using conventional paradigms due to the lack of supporting methods and tools.
- Although agents might be seen as AI technology, it is not wise to advertise them as such. Companies see AI as fuzzy, research oriented and not technology that can be used in practice. Therefore, in times of cost cutting they will not be willing to invest in AI technology.
- Educating companies in the use of agent technology is important but difficult if so many agent platforms, tools and methodologies exist in parallel.

The objective of the PROMAS TF meetings was to stimulate the discussion on the implementation issues of multi-agent systems, to make the existing works in this area visible to each other, and to create the possibility to establish cooperations between participants. In this respect, both meetings have been quite successful. Various talks from industry and academia were presented, discussion slots made it possible for participants to share their opinions and discuss their works, and participants indicated their interest for bi-lateral cooperations. We deliberately avoided a plan to establish a large-scale collaboration between the participants. Instead, we decided to leave the choice of possible collaborations to the participants themselves. Many interesting discussions, aside from the presented ones, appeared during the last two editions of PROMAS TF. We tried to gather as many discussed issues as possible into this report. Unfortunately, it was not possible to cover the entire discussion in detail.

Acknowledgements

We gratefully acknowledge the help and support of AgentLink III which provided the forum for much of the discussion used in this paper. We would also like to thank all attendees and invited speakers of the last two editions of PROMAS for their contributions and participation.

References

Arbab, F, 2004, Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science* **14**(3), 329–366.

https://doi.org/10.1017/S0269888905000482 Published online by Cambridge University Press

Arbab, F, Herman, I and Spilling, P, 1993, An overview of manifold and its implementation. *Concurrency: Practice and Experience* **5**(1), 23–70.

Baldoni, M, Boella, G and van der Torre, L, 2005, Bridging agent theory and object orientation: importing social roles in object oriented languages. In *Programming Multi-Agent Systems, Third International Workshop ProMAS 2005*.

Baldoni, M, Giordano, L, Martelli, A and Patti, V, 2000, Modeling agents in a logic action language. In *Proceeding of the Workshop on Practical Reasoning Agents, International Conference on Formal and Applied Practical Reasoning*.

Bordini, RH, Dastani, M, Dix, J and Fallah-Seghrouchni, AE (eds.), 2005a, *Programming Multi-Agent Systems, Second International Workshop ProMAS 2004* (*Lecture Notes in Computer Science, 3346*). Berlin: Springer.

Bordini, RH, Dastani, M, Dix, J and Fallah-Seghrouchni, AE (eds.), 2005b, *Multi-Agent Programming: Languages, Platforms and Applications* (*Multiagent Systems, Artificial Societies, and Simulated Organizations, 15*). Berlin: Springer.

Bordini, RH, Häubner, JF and Vieira, R, 2005c, Jason and the Golden Fleece of agent-oriented programming. In Bordini, RH, Dastani, M, Dix, J and Fallah-Seghrouchni, AE (eds.), *Multi-Agent Programming: Languages, Platforms and Applications* (*Multiagent Systems, Artificial Societies, and Simulated Organizations, 15*), ch. 1. Berlin: Springer, pp. 3–37.

Bresciani, P, Giorgini, P, Giunchiglia, F, Mylopoulos, J and Perini, A, 2003, TROPOS: an agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems* **8**(3), 203–236.

Brooks, RA, 1991, How to build complete creatures rather than isolated cognitive simulators. In VanLehn, K (ed.), *Architectures for Intelligence*. Hillsdale, NJ: Lawrence Erlbaum Associates, pp. 225–239.

Castelfranchi, C, 1990, Social power: a missed point in DAI, MAS and HCI. In Demazeau, Y and Muller, JP (eds.), *Decentralized A.I.: Proceedings of the First European Workshop on Modelling Autonomous Agents in a Multi- Agent World*. Amsterdam: North-Holland, pp. 49–62.

Ciampolini, A, Lamma, E, Mello, P, Toni, F and Torroni, P, 2003, Co-operation and competition in ALIAS: a logic framework for agents that negotiate. *Annals of Mathematics and Artificial Intelligence* **37**(1–2), 65–91.

Costantini, S and Tocchio, A, 2002, A logic programming language for multi-agent systems (*Lecture Notes in Artificial Intelligence, 2424*). Berlin: Springer, pp. 1–13.

Dastani, M, Hulstijn, J, Dignum, F and Meyer, J-JCh, 2004a, Issues in multiagent system development. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS'04)*.

Dastani, M, van Riemsdijk, MB, Dignum, F and Meyer, JJCh, 2004b, *A Programming Language for Cognitive Agents: Goal Directed 3APL* (*Lecture Notes in Artificial Intelligence, 3067*). Berlin: Springer, pp. 111–130.

Dastani, M, Dix, J and Fallah-Seghrouchni, AE (eds.), 2003, *Programming Multi-Agent Systems, First International Workshop, PROMAS 2003,* (*Lecture Notes in Computer Science, 3067*). Berlin: Springer.

De Giacomo, G, Lesperance, Y and Levesque, H, 2000, ConGolog 2000, a concurrent programming language based on the situation calculus. *Artificial Intelligence* **121**(1–2), 109–169.

Ferber, J and Gutknecht, O, 1998, A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS98)*.

Ferber, J, Gutknecht, O and Michel, F, 2003, *From Agents to Organizations: An Organizational View of Multi-agent Systems* (*Lecture Notes in Computer Science, 3067*). Berlin: Springer, pp. 214–230.

Fisher, M, 1993, *Concurrent METATEM—A Language for Modelling Reactive Systems* (*Lecture Notes in Computer Science, 694*). Berlin: Springer, pp. 185–196.

García-Serrano, A, Martinez, P and Teruel, D, 2001, Knowledge-modelling techniques in the e-commerce scenario. In *Proceedings of the Workshop on E-Business and the Intelligent Web, International Joint Conference on Artificial Intelligence*.

Gelernter, D, 1985, Generative communication in Linda. *ACM Transactions on Programming Languages and Systems* 7(1), 80–112.

Gomez-Sanz, J, Pavon, J and Garijo, F, 2005, Estimating costs for agent oriented software. In *Proceedings of the Sixth International Workshop on Agent-Oriented Software Engineering*.

Gomez-Sanz, J and Pavon, J, 2003, *Agent Oriented Software Engineering with INGENIAS* (*Lecture Notes in Computer Science, 2691*). Berlin: Springer, pp. 394–403.

Grossi, D, Dignum, F, Dastani, M and Royakkers, L, 2005, Foundations of organizational structures in multi-agent systems. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, Utrecht. New York: ACM Press, pp. 690–697.

Iglesias, CA, Garijo, M, Centeno-Gonzalez, J and Velasco, JR, 1998, *Analysis and Design of Multiagent Systems Using MAS-Common KADS* (*Lecture Notes in Computer Science, 1365*). London: Springer, pp. 313–327.

Ingrand, FF, Georgeff, MP and Rao, AS, 1992, An architecture for real-time reasoning and system control. *IEEE Expert, Knowledge-Based Diagnosis in Process Engineering* **7**(6), 34–44.

https://doi.org/10.1017/S0269888905000482 Published online by Cambridge University Press

Jayatilleke, G, Padgham, L and Winikoff, M, 2005, Component agent framework for non-experts (CAFnE) toolkit. *Software Agent-Based Applications, Platforms and Development Kits* (*Whitestein Series in Software Agent Technologies*). Birkhäuser Publishing Company.

Jennings, NR, 2000, On agent-based software engineering. *Artificial Intelligence* **117**(2), 277–296.

Jonker, CM and Treur, J, 1998, Compositional verification of multi-agent systems: a formal analysis of pro-activeness and reactiveness. *International Journal of Cooperative Information Systems* **11**(1–2), 51–92.

Kolp, M, Castro, J and Mylopoulos, J, 2002, Towards requirements-driven information systems engineering: the tropos project. *Information Systems* **27**(6), 365–389.

Laird, JE, Newell, A and Rosenbloom, PS, 1987, SOAR: an architecture for general intelligence. *Artificial Intelligence* **33**(1), 1–64.

Leite, JA, Alferes, JJ and Pereira, LM, 2002, *MINERVA: A Dynamic Logic Programming Agent Architecture* (*Lecture Notes in Artificial Intelligence, 2333*). Berlin: Springer, pp. 141–157.

Luck, M, McBurney, P and Preist, C, 2003, Agent technology: enabling next generation computing (a roadmap for agent based computing). Technical report, `http://www.agentlink.org/roadmap/index.html`.

Omicini, A, 2000, *SODA. Societies and Infrastructures in the Analysis and Design of Agent-based Systems* (*Lecture Notes in Computer Science, 1957*). Berlin: Springer, pp. 185–193.

Ossowski, S, Hernández, JZ, Belmonte, MV, Maseda, J, Fernández, A, García-Serrano, A, Triguero, F, Serrano, JM and Péerez-de-la-Cruz, JL, 2004, Multi-agent systems for decision support: a case study in the transportation management domain. *Applied Artificial Intelligence* **18**(9–10), 779–795.

Padgham, L and Winikoff, M, 2003, *Prometheus: A Methodology for Developing Intelligent Agents* (*Lecture Notes in Artificial Intelligence, 2585*). Berlin: Springer, pp. 174–185.

Penczek, W and Lomuscio, A, 2003, Verifying epistemic properties of multi-agent systems via bounded model checking. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*. New York: ACM Press, pp. 209–216.

Plotkin, GD, 1981, A structural approach to operational semantics. Technical report DAIMI FN-19, University of Aarhus.

Pokahr, A, Braubach, L and Lamersdorf, W, 2005, Jadex: a BDI reasoning engine. In Bordini, RH, Dastani, M, Dix, J and Fallah-Seghrouchni, AE (eds.), *Multi-Agent Programming: Languages, Platforms and Applications* (*Multiagent Systems, Artificial Societies, and Simulated Organizations, 15*), Berlin: Springer, ch. 6, pp. 149–174.

Ricci, A, Omicini, A and Denti, E, 2001, Enlightened agents in TuCSoN. *AI*IA Notizie* **XIV**, 48–49.

Ross, R, Collier, R and O'Hare G, 2005, *AF-APL: Bridging Principles and Practices in Agent Oriented Languages* (*Lecture Notes in Computer Science, 3346*). New York: Springer, pp. 66–88.

Serrano, JM and Ossowski, S, 2004, *On the Impact of Agent Communication Languages on the Implementation of Agent Systems* (*Lecture Notes in Computer Science, 3191*). Berlin: Springer, pp. 92–106.

Shoham, Y, 1993, Agent-oriented programming. *Artificial Intelligence* **60**, 51–92.

Sichman, JS, Conte, R, Castelfranchi, C and Demazeau, Y, 1994, A social reasoning mechanism based on dependence networks. *Proceedings of the Eleventh European Conference on Artificial Intelligence*. Chichester: John Wiley & Sons, pp. 188–192.

Stathis, K, Kakas, A, Lu, W, Demetriou, N, Endriss, U and Bracciali, A, 2004, PROSOCS: a platform for programming software agents in computational logic. In *Proceedings of the 4th International Symposium From Agent Theory to Agent Implementation (AT2AI-2004)*.

Thielscher, M, 2005, FLUX: a logic programming method for reasoning agents. *Journal of Theory and Practice of Logic Programming* **5**(4), 533–565.

van Aart, CJ, Van Marcke, K, Pels, RF and Smulders, JLFC, 2002, International insurance traffic with software agents. In *Proceedings of the Fifteenth European Conference on Artificial Intelligence (EACI2002)*.

van Aart, CJ, Wielinga, B and Schreiber, G, 2004, Organizational building blocks for design of distributed intelligent system. *International Journal of Human-Computer Studies* **61**(5), 567–599.

Winikoff, M, 2005, JACKtm intelligent agents: an industrial strength platform. In Bordini, RH, Dastani, M, Dix, J and Fallah-Seghrouchni, AE (eds.), *Multi-Agent Programming: Languages, Platforms and Applications* (*Multiagent Systems, Artificial Societies, and Simulated Organizations, 15*), Berlin: Springer, ch. 7, pp. 175–193.

Wooldridge, M, Fisher, M, Huget, MP and Parsons, S, 2002, Model checking multi-agent systems with Mable. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*. New York: ACM Press, pp. 952–959.

Wooldridge, M and Jennings, NR, 1995, Intelligent agents: theory and practice. *Knowledge Engineering Review* **10**(2), 115–152.

Zambonelli, F, Jennings, NR. &Wooldridge, M, 2003, Developing multiagent systems: the Gaia methodology. *ACM Transactions on Software Engineering and Methodology (TOSEM)* **12**(3), 317–370.

https://doi.org/10.1017/S0269888905000482 Published online by Cambridge University Press