Journal of Universal Computer Science, vol. 10, no. 4 (2004), 359-374
submitted: 17/10/03, accepted: 2/2/04, appeared: 28/4/04 © J.UCS

Methodologies for Developing Multi-Agent Systems

Jorge Gómez-Sanz
(Universidad Complutense Madrid, Spain
jjgomez@sip.ucm.es)
Juan Pavón
(Universidad Complutense Madrid, Spain
jpavon@sip.ucm.es)

Abstract: As agent technology has matured with the deployment of a variety of applications, particularly in open and dynamic environments such as the web, several methodologies and tools have been proposed to support software engineers during the development process of such systems. This article takes an overall look at representative agent-oriented methodologies by considering how they support specific agent-related concepts. This serves to identify areas in which this technology has shown its potential to solve new problems, e.g., the ability to manage complexity with an organizational perspective, goal-driven modelling as a way to build robust behaviors for adaptive systems, or the definition of notation and mechanisms to implement high-level interactions and protocols between agents. In order to be fully applicable, the challenge today is the maturity of supporting tools, and new methods for validation and verification of multi-agent systems.

Key Words: software agents, intelligent agents, multi-agent systems, agent-oriented software engineering, agent-oriented methodologies
Category: D.1, D.2, I.2.11

1 Introduction

Until recently, the development of multi-agent systems (MAS) has been more of an art than a structured discipline, and the success of the technology relied on specific applications, especially for adaptive and collaborative systems. The generalization of applications and the study of engineering issues have contributed to the agent concept being considered as a new abstraction, which can be applied, throughout the whole software life-cycle, for building new kinds of services in open and dynamic environments. In this paper, we review methods and tools for the development of agent-based applications from this perspective, and discuss how agent-related concepts can support the software engineering activities to tackle the complexity of future software systems.

In [Zambonelli and Parunak, 2002], the authors argue that today's software systems are reaching greater degrees of complexity in several aspects, not only size, as other factors get involved. First, there is a trend to provide new services in open environments, such as web services. This means that new system elements have to be able to acquire knowledge about their context and interact with other entities. Besides, their context changes over time as other elements appear, disappear, or modify their behavior (in the case of processes) or contents (in the case of information). Different systems, not

necessarily software systems, co-exist in the same environment, either collaborating or competing. Sharing the environment implies that their actions will try to change this environment, perhaps at the same time. Therefore, it is not possible to assure that an action will have the expected result. In these situations, traditional control structures or conventional synchronization mechanisms are not always valid because of the dynamics of the environment. Agents have different techniques to deal with the uncertainty of system dynamics, such as learning capabilities or planning capabilities. This way, an agent can detect that a task is not performing as it should, and replace it with another task or decide to collaborate with other agents.

Another characteristic of this open environment is heterogeneity: multiple computing devices everywhere, with different capabilities, and connected to (more and more wireless) networks. This implies higher degrees of distribution in the management of entities, in the location of control, and in the interactions. The agent approach assumes these considerations at its foundations since agents are conceived as autonomous entities that can reside in a node of a network, or even migrate in the case of mobile agents.

From the perspective of knowledge processing and management, there is an increasing need for processing information data to provide knowledge-based services. At this point, new mechanisms for information processing are required, and interaction among system components requires a higher level of abstraction, with more support for semantic processing. The use of ontologies and agent communication languages is advantageous regarding traditional object-oriented communication mechanisms, which support syntactic interoperability only.

Finally, the usability of computer-based systems has increased as more people, with a great variety of profiles, use them. Higher degrees of personalization have become important for service acceptance and differentiation. This means highly reconfigurable systems in which special processing is required for each user. This is often addressed by considering one agent as a personal assistant for each user, with capabilities to learn and adapt to the user's changing profile.

It seems necessary to justify why objects are not enough to build such systems. Although many MASs are implemented with object-oriented programming languages and tools, agents are quite different from objects. Firstly, in their degree of autonomy [Wooldridge and Ciancarini, 2000]. An object may exhibit autonomy over its state (by restricting access to instance variables so that only the object can control them) but not over its behavior, as the object must execute methods invoked on it (the execution of methods on an object is mainly determined by external entities). In terms of agents, responsibilities are clearly separated from one agent to another, and these are characterized in terms of goals rather than as a set of functions. Agents are autonomous to decide which behavior is more convenient by taking their context and their goals into account. When an agent receives a request to perform an action, it will consider whether to execute or refuse the task. This is important from a software engineering perspective since goals are considered more stable than input-output relationships (functional approach)

when a system evolves, and this is where contributions from the field of artificial intelligence come into play to model agents and their social behavior. Given that an agent is a goal-based entity, its behavior can be conceived as a reasoning system in which decisions on which task to execute depend on the current knowledge of the environment, the status of goal achievement, and actual capabilities of an agent and its neighbors.

From an engineering viewpoint, agents can also be considered as an extension of the component model. Agents can be deployed in a distributed system quite easily, and they can be configured not only with specific values for some parameters, but also behavioristically. Taken to an extreme, agents can learn new procedures, and even new interaction languages and protocols. A MAS represents then a set of highly configurable entities, which can also be reused as an organization. New systems can be conceived as a combination of agent organizations, each one providing services and relying on services of other organizations. In this respect, the agent paradigm provides for both the horizontal and vertical breakdown of complex system development. Because of the growing possibilities of such an approach, work on coordinating agent systems is considered to be fundamental.

These ideas need to be organized methodologically so that they can be incorporated into current software engineering practices. In the rest of this paper, we review how agent-related concepts can be applied to define agent-oriented methodologies for the development of complex systems. The organizational view of a MAS can help to deal with such complexity: first, by separating the modelling of the system into several views [see Section 2], which is illustrated with AAII/BDI, Vowels Engineering, CoMoMAS, MAS-CommonKADS, MESSAGE, INGENIAS, MASSIVE, and ODAC; second, by providing a framework to introduce other concepts such as roles, services, and interactions [see Section 3], which is illustrated with Gaia, MaSE, and AALAADIN. The experience with these methodologies has contributed to mature agent-related concepts and there is currently a trend to define a common modelling language as an extension to UML [see Section 4]. The survey finishes by reviewing verification techniques [see Section 5] and implementation tools [see Section 6]. In the final section, we pose the question of what methodology to choose and conclude that this is still open to discussion, although some authors have started to propose evaluation frameworks [see Section 7].

2 Different Views of a Multi-Agent System

The modelling of a MAS should be considered from different complementary viewpoints to deal with its complexity. One of the first proposals, the AAII/BDI methodology [Kinny et al., 1996], considers two viewpoints: external and internal. The external viewpoint considers agents as complex objects (with their own purposes, responsibilities, services, and information), and external interactions, which is consistent with the classical view of agents as autonomous entities that interact with their environment. The

internal viewpoint considers the elements required by the particular agent architecture, e.g., a set of beliefs, goals and plans. This methodology is guided by the elaboration and refinement of the models for each view: first the external viewpoint is considered; then, the internal viewpoint; later, external models are fed back, and the process continues until enough implementation details are obtained.

The purpose of the external viewpoint is to identify an agent class hierarchy (the agent model) and a set of relationships between agents (the interaction model). They are constructed in four steps, namely:

1. Identification of roles in the application domain.
2. For each role, identify its associated responsibilities, and the services provided and used to fulfil those responsibilities.
3. For each service, identify the interactions associated with the provision of the service. This allows to determine control relationships between agents.
4. Refine the agent hierarchy, e.g., refactoring, composition, and aggregation.

This process leads to an assignment of functionality (services) to agents, and associations (services relationships and interactions) between them.

In this methodology, the internal viewpoint is highly dependent on the BDI architecture [Rao and Georgeff, 1991] since agents have some mental attitudes called beliefs, desires, and intentions, i.e., agents have a mental state that consists of informational, motivational, and deliberative states respectively. Beliefs represent the information about the environment, the internal state the agent may hold, and the actions it may perform (belief model); the agent will try to achieve a set of goals, and will respond to certain events (goal model); the control structure of the agent is defined in terms of plans (plan model). The methodology for the development of these models begins by considering the services provided by the agent and the associated events and interactions. They determine the goals, and the analysis consists of breaking them down into subgoals, which leads to the identification of plans. This is summarized in two steps:

1. Analyzing the means of achieving the goals. This consists of a breakdown of the goal into subgoals and actions, for the different contexts in which the goal has to be achieved. This process is applied repeatedly to subgoals.
2. Build the beliefs of the system, by analyzing the context and conditions that control the execution of activities.

Note that the emphasis is on goals instead of behaviors since they are considered more stable in general. Thus, the resulting design is more stable, robust, and modular. If the context changes, plans for new contexts can be added without changing plans for the same goal.

Most of the methodologies take more viewpoints into account. Vowels Engineering [Ricordel and Demazeau, 2002] proposes five, which correspond to the Latin vowels: Agent, Environment, Interactions, Organization, and User. Different techniques can be applied to analyze and design each aspect. Agents can be conceived as simple automata or complex knowledge-based systems. Interactions can be studied as physical models, e.g., wavelength propagation, or as speech acts. Organizations can be inspired in biological models or ruled by sociological models. The purpose of this methodology is to consider component libraries that provide solutions for each aspect, so that the designer can instantiate an agent model, an organization model, and so on. The methodology proposes to consider vowels (aspects) in a certain order, depending on the kind of system being developed. For instance, if social relationships are important, the development process should start with the organization. If the process starts with agents, then the system will have an organization that probably emerges as a result of the interactions of individual agents. This methodology is currently supported by the Volcano component-oriented platform.

Structuring of the system into viewpoints was already applied in a methodology for knowledge engineering called CommonKADS [Schreiber et al., 1994]. It proposed six models to identify the organization in which the knowledge base system (KBS) works, tasks, agents, e.g., the expert system, communications (mainly between the agent and the user), experience (domain knowledge, and resolution knowledge), and design (architecture of the KBS). As expert systems are quite centralized, this model needed extensions to manage the distributed nature of MAS. This is the purpose of the refinements proposed in CoMoMas [Glaser, 1996] for the agent model with social, cooperative and cognitive aspects, and adding a cooperative model and a system model to consider the organizational aspects of the MAS. MAS-CommonKADS [Iglesias et al., 1998] is more relevant since it uses the OMT object-oriented notation to structure systems, use cases to capture requirements, and standard protocol specification techniques such as SDL [ITU-T, 1999] and message sequence charts to describe agent interactions. There is a case study developed with MAS-CommonKADS [Arenas and Barrera-Sanabria, 2002] and the authors plan to announce some supporting tools soon.

MESSAGE [Caire et al., 2001] is a recent proposal to integrate different methodologies. It builds on five viewpoints that are described with meta-models as UML extensions [Gómez-Sanz et al., 2002]. A set of development tools for analysis, design, code generation and validation are available and they build on these meta-models in INGENIAS [Pavón and Gómez-Sanz, 2003], a refinement and extension of MESSAGE. The development of a MAS consists of identifying elements for each viewpoint and then performing a set of activities that are defined in the context of the Unified Process [Jacobson et al., 1999]. The implementation can be generated automatically for different target platforms with the INGENIAS Development Kit. The proposed viewpoints are the following:

- The agent viewpoint, which describes an agent's responsibilities with tasks and roles. It also takes into account the control of the agent and defines its goals and the mental states required during execution.
- The organization viewpoint, which determines the architecture of a system. Structural relationships are not restricted to hierarchies between roles. These structures are delegated to specialized entities called groups. In the organization model there are also power relationships among groups, organizations, and agents. The functionality of the organization is expressed using workflows, which show consumer/producer associations between tasks as well as the assignment of responsibilities for their execution, and resources associated to each.
- The environment viewpoint, which defines the sensors and effectors of the agents. It also identifies available resources as well as already existing agents and applications, e.g., legacy systems, databases, web information systems.
- The tasks and goals viewpoint, which is strongly influenced by the BDI model and Newell's principle of rationality. Its main purpose is to justify the execution of tasks in terms of the goals. It also provides the breakdown of tasks and goals. To relate both, there are specialized relationships that detail which information is needed to consider a goal solved or failed. Finally, this viewpoint also provides low-level details of tasks in the system, and describes which resources are needed during an execution, which software modules are used throughout the process, and which are the inputs and outputs.
- The interaction viewpoint, which describes how coordination among agents takes place. It goes a step further than UML sequence diagrams since it reflects the motivation of the interaction and its participants. It also includes information about the mental state required in each agent throughout the interaction as well as tasks executed in the process. This allows us to justify at a design level why an agent engages in a interaction and why it should continue.

Other methodologies that emphasize the modelling of the MAS from different viewpoints are MASSIVE [Lind, 2000], which proposes seven viewpoints (environment, task, role, interaction, society, architectural, and system), and ODAC [Gervais, 2003], which uses the five ODP viewpoints (enterprise, information, computational, technology and engineering) [X.900, 1995].

3 Roles, Services, Interactions, and Organizations

From the preceding section, it is clear that roles, services and interactions drive the modelling of a MAS. The concept of role appears naturally when adopting an organizational view of the system as in the case of MAS. Roles identify functionality, in terms

of services, and identify characteristics of parties in interactions. When instantiated, roles are played by agents, which are supposed to have the capabilities to perform the corresponding services.

Apart from AAII/BDI, these concepts have been fully developed in methodologies such as Gaia and MaSE. The Gaia methodology [Wooldridge et al., 2000] addresses the analysis of agent-based systems without referencing implementation issues. This is achieved by considering the system as a society or organization. The organization consists of a collection of roles, that have relationships with one another, and that take part in institutionalized patterns of interactions with other roles. Each role is defined by four attributes: responsibilities (its functionality, described as liveness and safety properties), permissions (in terms of rights, identify the resources that are available to the role, such as information resources), activities (those computations that can be performed without interacting with others), and protocols (the way that it can interact with other roles). These protocols are further defined in the interaction model. The analysis consists of iterating repeatedly on the following steps:

1. Identify the roles in the system, as individuals, departments or organizations. The result is a list of roles with an informal description.

2. For each role, identify and document the associated protocols. This gives an interaction model.

3. Elaborate the roles model with the previous information.

Design in Gaia produces three models: an agent model, which identifies agent types (essentially, as aggregation of roles) and their instances in a system, a services model (functions of each agent), and an acquaintance model, a directed graph that simply describes the communication links between agents.

Gaia does not attempt to provide a computational model of the agent system, but to describe how a society of agents cooperate to achieve the system-level goals, and what is required by each individual agent in order to do this. Therefore, after applying Gaia, the developer has to use other design techniques to accomplish an implementable system. In this sense, Gaia is quite limited, and its relevance for agent-oriented software engineering comes from its influence on other methodologies, particularly in the analysis of roles and interactions.

MaSE (Multi-agent systems Software Engineering) [DeLoach et al., 2001], on the contrary, supports the whole development life-cycle, from problem description to realization. MaSE adopts the object-oriented paradigm (UML), by considering agents as specialized proactive objects that coordinate by means of conversations. The development process in MaSE consists of a collection of steps, most of them supported by the agentTool system [DeLoach, 2001]. The first step is to capture system goals from user requirements, and structuring them into a goal hierarchy. This is followed by use case analysis and the definition of the corresponding sequence diagrams. From these

diagrams, it is possible to derive roles and their associated tasks. Roles in MaSE form the foundation for agent class definition and represent system goals during the design phase. The design phase in MaSE produces an agent class diagram, by assigning roles to specific agent classes, the conversations between agents, the design of internal agent architectures, and the deployment of agents in a system. A conversation is a coordination protocol between two agents, and it is described with two finite state machines, one for each party (initiator and responder).

Roles and services help to structure the functionality associated to an agent or a group of agents, contributing to the understandability and manageability of complex systems, with an organizational perspective. This is the starting point of AALAADIN project [Ferber and Gutknecht, 1998], whose goal is to provide tools to analyze, design, formalize and develop multi-agent systems from an organizational perspective. This project has developed a generic meta-model of MAS based on organizational concepts of agents, groups, and roles. This model is called AGR (Agent/Group/Role), and it is supported by MadKit [MADKIT, 1999]. This model was extended in MESSAGE and INGENIAS, which developed the organization concept to consider different contexts in which relationships and interactions between agents or roles may take place [Garijo et al., 2000]. Initially inspired by the AALAADIN approach, they added some extra concepts: workflows, resources, and their integration in the MAS specification. Workflows and interactions are complementary concepts in INGENIAS. Workflows describe dependencies of tasks, the agents or roles that are responsible for their execution, and which flows of information exist. Interactions define the exchange of messages and the timing in the execution of tasks, as well as the conditions to meet in order to continue an interaction.

These methodologies highlight organizations as a key element to study MAS and consider them as something more than a set of roles and dependencies. An organization corresponds to the system architecture since it defines the scope of agents and roles, provided services, pursued goals, tasks to be executed, and available resources.

4 Towards a Unified Notation for MAS

As it has been recognized within the Special Interest Group on Methodologies and Software Engineering for Agent Systems (MSEAS) at AgentLink, the agent community needs to agree on concepts and vocabulary to ease the comparison of existing methodologies and provide a solid foundations for the evolution of agent development methods and tools [Zambonelli et al., 2002].

Some methodologies are based on the definition of meta-models, which simplifies the integration of new concepts or the modification of existing ones, e.g., INGENIAS [Gómez-Sanz et al., 2002], AALAADIN [Ferber and Gutknecht, 1998]. In contrast, some methodologies describe their processes by means of meta-models, e.g., ADELFE [Picard et al., 2002], which extends the Unified Process by adding new ac-

tivities, e.g., characterization of the environment, verification of the MAS, and identification of agents. It is supported by OpenTool to edit AUML diagrams, and a library of reusable components.

The trend that is gaining more importance seems to be the design of a unified notation based on extending UML with agent-specific features that are not covered by versions 1.4 or 2.0. The result is called Agent-UML (AUML) [AUML Team, 03]. This is currently being adopted by FIPA, the main international organization for agent standards. AUML started by extending UML to specify agent interaction protocols [Odell et al., 2001]. Protocol diagrams in AUML extend UML sequence diagrams by providing mechanisms to define agent roles, agent lifelines (interaction threads, which can split into two or more lifelines and merge at some subsequent point), nested and interleaved protocols (patterns of interaction that can be reused with guards and constraints), and extended semantics for UML messages (for instance, to indicate the associated communicative act, whether messages are synchronous or asynchronous, and other characteristics such as blocking, non-blocking and time-constrains). These diagrams have been used to specify FIPA interaction protocols.

Another extension to UML can be found in [Parunak and Odell, 2002]. It brings together several agent organization-related concepts such as roles, group, dependencies and speech acts, whose relevance to agent-based development has been discussed in the previous sections. This approach is based on using three artifacts, namely:

1. Group, as a set of agents that share common interests, purposes or tasks. A group is modelled by a class diagram and swinlanes to organize the roles in the group.

2. Role, as a representation of an agent's function. One agent can play several roles at a time, even in different organizations. The relationships between organizations, agents and roles can also be depicted in a class diagram.

3. Environment, which provides three data processing functions: it merges information from different agents that come to the same location at different times, it distributes data from one location to nearby locations, and it provides truth maintenance by forgetting data that become obsolete.

Currently, the FIPA Modelling Technical Committee is also addressing the modelling of agent classes, based on the proposal by [Bauer, 2002]. They rely on the assumption that agent autonomy, proactivity, reactivity, and speech-act based communications require additional features to represent the internal state of the agents, other than those used for objects. Here, an agent consists of a communicator, which performs physical communication, a head, which deals with goals or states, and a body, which performs the actions. The proposed notation allows to define agent classes, agent classes satisfying distinguished roles, and agent instances. An agent class is defined by:

1. An agent class name and, optionally, a list of distinguished roles.

2. A state description, which looks similar to a field description in class diagrams, but allows to express well-formed formulae for logical descriptions of the state. For instance, this could be useful for defining the beliefs, desires, intentions, and goals of an agent.
3. Actions, which can be denoted by the pro-active and re-active stereotypes.
4. Methods, as in UML classes.
5. Capabilities of the agent, e.g., FIPA service descriptions.
6. Communicative acts that define the main interface of the agent. Communicative acts are themselves defined as other classes.

Other aspects of agent modelling are currently a subject of research, but should be included in the FIPA AUML specifications. The evolution of this work can be tracked on-line at `www.auml.org`.

5 Verification and Validation of MAS

Research in MAS verification is based on a trend to formalize agent systems. Formal methods applied in MASs rely on existing techniques based on axiomatic approaches and semantic-based approaches [Wooldridge and Ciancarini, 2000].

Axiomatic-based approaches propose proofs in the form of automatic theorem proving, which can sometimes determine if a specification satisfies a model. Well-known examples of such an approach include ConGolog [Giacomo et al., 2000] as well as CASL [Shapiro et al., 2002]. For an overview of automated theorem proving procedures consult [Bledsoe, 1985]. Since theorem proving techniques have a high computational cost and may not be decidable, researchers tend to focus on semantic-based approaches. Such approaches take the semantics of a language into consideration to discern whether a formula is true or not in a concrete model. Model checking plays a significant role here, but this is closer to testing than to verifying since it deals with software, not only with formal specifications [Chandra et al., 2002]. However, model checking is not usually interpreted this way in this context since programs are not written in imperative languages, but in declarative modal languages. [Bordini et al., 2003] and previous papers report on how to apply model checking to BDI based systems defined with AgentSpeak(L) [Rao, 1996]. This work was previously initiated in the context of the AAII/BDI methodology. In [Penczek and Lomuscio, 2003], the author also studies model checking to verify the knowledge of agents in a sample scenario in which two generals have to coordinate an attack. These approaches can be considered representative since both start from a formal specification of the system. Just to provide a different point of view, we would like to mention [Fuentes et al., 2003], which proposes a new way of model checking using social theories as a model. Even if they are not so formal, they are closer to standard human thinking.

Validation, on the other hand, depends on having initial requirements and checking if these requirements hold in the final system. This task refers to capturing initial requirements and expressing them. There is an important amount of work in requirements elicitation in the agent domain that is considered relevant in the software engineering community [Dardenne et al., 1993]. KAOS defines a meta-model of tasks and goals to capture requirements and provides a tool called GRAIL/KAOS to represent them [Darimont et al., 1997]. The conceptual framework called i* [Yu, 1997] defines actors, beliefs, commitments, and goals to model organizational environments and their information systems. Recently, i* has been adopted as an underlying framework for an AOSE methodology named Tropos [Mylopoulos and Castro, 2000]. Tropos adds a development process and automated translation methods from an i* specification into agents supported by the Jack platform [Busetta et al., 1999]. The approach of the specification language Albert II is more formal [Heymans and Dubois, 1998]. The paper describes a tool able to simulate a specification so that clients can see how it should work and perform the validation themselves; there is also an example of validation.

Some of the existing methodologies apply these results, but it is not common. MaSE uses model checking to detect deadlocks in communication among agents. It uses SPIN [Holzmann, 1991], a well-known model checking tool that requires the translation of a MaSE specification into the PROMELA language. DESIRE, a component-based framework, proposes a compositional verification method in which the authors suggest to correlate the verification of system properties with the properties of its subcomponents [Brazier et al., 1997]. The properties to be verified depend on the domain. In [Brazier et al., 1994], the domain is agent negotiation in load balance. The authors suggest verification of termination, i.e., a negotiation terminates if there is a time after which no bids are made, and communication groundedness, i.e., an agent can hear what is said during a negotiation. Finally, INGENIAS uses Activity Theory [Leontiev, 1978] to find contradiction patterns [Fuentes et al., 2003]. This theory provides explanations of how human societies function using informal models. [Fuentes et al., 2003] shows how these models can be formalized and applied to develop a MAS.

6 Implementation

Experience in the application of methodologies, particularly those with tool support, has demonstrated the importance of tools to control the development process in all phases and support developers to produce and measure the quality of the results according to the methodology. Traditionally, the tools in this domain are composed of a GUI front-end that helps configuring an existing framework. Although they can generate functional systems, sometimes they are highly coupled to a specific framework, e.g., Zeus [ZEUS, 1999], and modifying it is difficult. MaSE generates code for different frameworks, but code generation facilities are not totally decoupled. Therefore, adapting MaSE to another framework is still not trivial, but easier than Zeus. Currently,

there are new tools that are based on the definition of meta-models. This includes PASSI [Burrafato and Cossentino, 2002], which is integrated in Rational Rose, and the INGENIAS Development Kit [GRASIA! research group, 2003], which supports both editing agent based specifications and code generation for several target platforms. Thanks to meta-models, these tools are more adequate for industrial development since they can be adapted to a variety of frameworks.

Besides tools, agents can be developed by reusing existing software components from agent platforms and specialized libraries. With regard to platforms, JADE is currently the most used FIPA-compliant platform [Bellifemine et al., 2001][FIPA, 2003]. It provides some building blocks for agent communications, and utility agents for remote monitoring of life-cycles and communications. Regarding the world of mobile agents, Grasshopper [IKV++ Technologies AG, 1998] follows the corresponding OMG standard, MASIF [OMG, 1999]. These platforms provide a basic agent class that provides access to the platform services, which range from agent management (creation, destruction, and monitoring) to mobility. However, the internals of the agent, the decision mechanisms, learning capabilities, or social abilities, need to be implemented by the developer.

To cover missing features, developers need to use third-party libraries and frameworks. Examples include SOAR [Laird et al., 1999], the Cougaar agent architectures [DARPA, 2003], the WEKA library [University of Waikato, 2004], and the JESS engine [Friedman-Hill, 2003]. SOAR provides a deliberative architecture, which is derived from the original results in [Laird et al., 1987]. There have been applications of SOAR ranging from modelling human behavior in urban combat to players in first-person-shoot'em-up games. Cougaar, according to their experiments, may be the most stable agent architecture today. JESS is a Java implementation of CLIPS [NASA, 2003], and it is usually applied to define behavior rules of agents. WEKA includes facilities to define data-mining and machine learning capabilities.

For an exhaustive list of agent-related resources, we suggest that the reader should consult [Mangina, 2002].

7 Conclusions

Given the diversity of agent-based methodologies, which one should you use? The answer, of course, is not simple. If the developer is familiar with knowledge-based systems and has worked with the CommonKADS methodology, it would be easy to adopt MAS-CommonKADS. For an object-oriented software developer, a methodology such as MaSE or Adelfe does not require too many changes, since they are based on the Unified Process. However, if the developer wants to fully exploit agent concepts, it is better to seek for more agent-oriented approaches, e.g., Zeus or INGENIAS.

Although some comparison frameworks are clearly biased by its authors' background on MAS, the evaluation of methodologies has gained importance recently. One

of the very first evaluations refers to agent development platforms, rather than methodologies [Ricordel and Demazeau, 2000], but it takes into account their support in analysis, design, development, and deployment. In [Shehory and Sturm, 2001], they pay attention to modelling aspects, but do not take into account the development process. In [Cernuzzi and G. Rossi, 2002], the authors go a step forward with a quantitative approach in which they give numerical estimations on the extent to which a methodology covers a specific feature. Last, but not least, [O'Malley and DeLoach, 2001] identifies a collection of criteria to decide what methodology to choose. It considers the management and needs that are required by each type of project. Although this is applied to MaSE, the criteria they identify can be adapted to other methodologies.

We have created a web site to discuss this issue, and we provide a set of criteria and case studies that can be applied to test and evaluate agent-oriented methodologies. The site is available at `http://ma.ei.uvigo.es/aose/`.

Acknowledgements

This work was supported by the Spanish Ministry of Science and Technology under grants TIC2002-04516-C03-03 and TIC2001-5108-E.

References

[Arenas and Barrera-Sanabria, 2002] Arenas, A. and Barrera-Sanabria, G. (2002). Applying the MAS-CommonKADS methodology to the flights reservation problem: Integrating coordination and expertise. In *The 5th Joint Conference on Knowledge-Based Software Engineering*.

[AUML Team, 03] AUML Team (03). Agent UML web site. `http://www.auml.org`.

[Bauer, 2002] Bauer, B. (2002). UML class diagrams: Revisited in the context of agent-based systems. In Wooldridge, M., Weiß, G., and Cianciarini, P., editors, *Agent-Oriented Software Engineering II: Second International Workshop*, volume 2222 of *LNCS*, pages 1–8. Springer-Verlag.

[Bellifemine et al., 2001] Bellifemine, F., Poggi, A., and Rimassa, G. (2001). JADE: a FIPA 2000 compliant agent development environment. In *Proc. of the 5th International Conference on Autonomous Agents*, pages 216–217. ACM Press.

[Bledsoe, 1985] Bledsoe, L. J. H. W. (1985). What is automated theorem proving? *Journal of Automated Reasoning*, 1(1):23–28.

[Bordini et al., 2003] Bordini, R., Fisher, M., Visser, W., and Wooldridge, M. (2003). Verifiable multi-agent programs. In *Proc. of the First International Workshop on Programming Multiagent Systems*, pages 1045–1052. Springer Verlag.

[Brazier et al., 1994] Brazier, F., van Langen, P., Treur, J., Wijngaards, N., and Willems, M. (1994). Modelling a design task in DESIRE: the VT example. Technical Report IR-377, Universiteit Amsterdam, Department of Mathematics and Computer Science, Vrije, Amsterdam.

[Brazier et al., 1997] Brazier, F. T., Dunin-Keplicz, B., Jennings, N., and Treur, J. (1997). DESIRE: Modelling multi-agent systems in a compositional framework. *International Journal of Cooperative Information Systems*, 6(1):67–94.

[Burrafato and Cossentino, 2002] Burrafato, P. and Cossentino, M. (2002). Designing a multi-agent solution for a bookstore with the PASSI methodology. In *4th International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002)*, Toronto, Ontario, Canada. CEUR-WS.

[Busetta et al., 1999] Busetta, P., Rönnquist, R., Hodgson, A., and Lucas, A. (1999). JACK Intelligent Agents – Components for intelligent agents in Java. *Agentlink News*, 2:2–5.

[Caire et al., 2001] Caire, G., Leal, F., Chainho, P., Evans, R., Garijo, F., Gómez-Sanz, J., Pavón, J., Kerney, P., Stark, J., and Massonet, P. (2001). Agent oriented analysis using MESSAGE/UML. In Wooldridge, M., Weiß, G., and Cianciarini, P., editors, *Agent-Oriented Software Engineering II: 2nd International Workshop*, LNCS 2222, pages 119–135. Springer Verlag.

[Cernuzzi and G. Rossi, 2002] Cernuzzi, L. and G. Rossi, G. (2002). On the evaluation of agent oriented methodologies. In *Proc. of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies*. COTAR.

[Chandra et al., 2002] Chandra, D., Godefroid, P., and Palm, C. (2002). Software model checking in practice: an industrial case study. In *Proc. ICSE'02*, pages 431–441. ACM Press.

[Dardenne et al., 1993] Dardenne, A., van Lamsweerde, A., and Fickas, S. (1993). Goal-directed requirements acquisition. *Science of Computer Programming*, 20:3–50.

[Darimont et al., 1997] Darimont, R., Delor, E., Massonet, P., and van Lamsweerde, A. (1997). GRAIL/KAOS: an environment for goal-driven requirements engineering. In *Proc. ICSE'97*, pages 612–613. ACM Press.

[DARPA, 2003] DARPA (2003). Cognitive agent architecture. `http://www.cougaar.org`.

[DeLoach, 2001] DeLoach, S. (2001). Analysis and Design using MaSE and agenTool. In *Proc. of the 12th Midwest Artificial Intelligence and Cognitive Science Conferece (MAICS)*, Miami University. Miami University Press.

[DeLoach et al., 2001] DeLoach, S., Wood, M., and Sparkman, C. (2001). Multiagent systems engineering. *Int. Journal of Software Engineering and Knowledge Engineering*, 11(3):231–258.

[Ferber and Gutknecht, 1998] Ferber, J. and Gutknecht, O. (1998). A meta-model for the analysis and design of organizations in multi-agent systems. In *Proc. ICMAS'98*, pages 128–135.

[FIPA, 2003] FIPA (2003). Foundation for Intelligent Agents. `http://www.fipa.org`.

[Friedman-Hill, 2003] Friedman-Hill, E. (2003). Java expert system shell (JESS). `http://herzberg.ca.sandia.gov/jess`.

[Fuentes et al., 2003] Fuentes, R., Gómez-Sanz, J., and Pavón, J. (2003). Activity theory for the analysis and design of multi-agent systems. In *Agent-Oriented Software Engineering IV*, pages 110 – 122. Springer Verlag.

[Garijo et al., 2000] Garijo, F., Gómez-Sanz, J., Pavón, J., and Massonet, P. (2000). Multi-agent system organization. An engineering perspective. In *10th European Workshop on Modelling Autonomous Agents in a Multi-Agent World*.

[Gervais, 2003] Gervais, M. (2003). ODAC: An agent-oriented methodology based on ODP. *Journal of Autonomous Agents and Multi-Agent Systems*, 7(3):199–228.

[Giacomo et al., 2000] Giacomo, G. D., Lesperance, Y., and Levesque, H. J. (2000). Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121:109–169.

[Glaser, 1996] Glaser, N. (1996). *Contribution to Knowledge Modelling in a Multi-Agent Framework (the CoMoMAS Approach)*. Ph.d. thesis, L'Universtité Henri Poincaré, Nancy I, France.

[Gómez-Sanz et al., 2002] Gómez-Sanz, J., Pavón, J., and Garijo, F. (2002). Meta-models for building multi-agent systems. In *Proc. ACM SAC'02*, pages 37–41. ACM Press.

[GRASIA! research group, 2003] GRASIA! research group (2003). INGENIAS IDK. `http://ingenias.sourceforge.net`.

[Heymans and Dubois, 1998] Heymans, P. and Dubois, E. (1998). Scenario-based techniques for supporting the elaboration and the validation of formal requirements. Technical Report CREWS Report 98-15, Universite de Namur, Belgium.

[Holzmann, 1991] Holzmann, G. (1991). *Design and Validation of Computer Protocols*. Prentice-Hall, Englewood Cliffs, New Jersey.

[Iglesias et al., 1998] Iglesias, C., Garijo, M., Gonzales, J., and Velasco, J. R. (1998). Analysis and design of multi-agent systems using MAS-CommonKADS. In Singh, M., Rao, A., and

Wooldridge, M., editors, *Intelligent Agents IV. Proc. of the Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL-97)*, volume 1365 of *LNAI*, pages 313–326. Springer-Verlag.

[IKV++ Technologies AG, 1998] IKV++ Technologies AG (1998). Grasshopper. `http://www.grasshopper.de/index.html`.

[ITU-T, 1999] ITU-T (1999). Recommendation Z.100: Specification and Description Language (SDL).

[Jacobson et al., 1999] Jacobson, I., Rumbaugh, J., and Booch, G. (1999). *The Unified Software Development Process*. Addison-Wesley.

[Kinny et al., 1996] Kinny, D., Georgeff, M., and Rao, A. (1996). A methodology and modelling technique for systems of BDI agents. In van der Velde, W. and Perram, J., editors, *Agents Breaking Away: Proc. of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-96)*, volume 1038 of *LNAI*, pages 56–71. Springer-Verlag.

[Laird et al., 1999] Laird, J., Congdom, C. B., and Coulter, K. (1999). *The SOAR's users manual v.8.2*. The Soar Group, Artificial Intelligence Laboratory, University of Michigan.

[Laird et al., 1987] Laird, J., Newell, A., and Rosenbloom, P. (1987). SOAR: an architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64.

[Leontiev, 1978] Leontiev, A. (1978). *Activity, Consciousness, and Personality*. Prentice Hall.

[Lind, 2000] Lind, J. (2000). *MASSIVE: Software Engineering for Multiagent Systems*. Ph.d. thesis, University of the Saarland.

[MADKIT, 1999] MADKIT (1999). *Multi-Agent Development KIT*.

[Mangina, 2002] Mangina, E. (2002). Review of software products for multi-agent systems. survey, AgentLink.

[Mylopoulos and Castro, 2000] Mylopoulos, J. and Castro, J. (2000). Tropos: A framework for requirements-driven software development. In *Proc. CAISE'00*.

[NASA, 2003] NASA (2003). C language integrated production system (CLIPS). `http://www.ghgcorp.com/clips/CLIPS.html`.

[Odell et al., 2001] Odell, J., Parunak, V., and Bauer, B. (2001). Representing agent interaction protocols in UML. In Ciancarini, P. and Wooldridge, M., editors, *Agent-oriented software engineering. Proceedings of the First International Workshop (AOSE-2000)*, volume 1957 of *LNAI*, pages 121–140. Springer-Verlag.

[O'Malley and DeLoach, 2001] O'Malley, S. and DeLoach, S. (2001). Determining when to use an agent-oriented software engineering paradigm. In Wooldridge, M., Weiß, G., and Ciancarini, P., editors, *Agent-Oriented Software Engineering II*, volume 2222 of *LNCS*, pages 188–205, 2nd International Workshop, AOSE 2001, Montreal, Canada. Springer Verlag.

[OMG, 1999] OMG (1999). Mobile Agent System Interoperability Facility (MASIF). `http://www.fokus.gmd.de/research/cc/ecco/masif`.

[Parunak and Odell, 2002] Parunak, H. V. D. and Odell, J. (2002). Represening social structures in UML. In Wooldridge, M., Weiß, G., and Cianciarini, P., editors, *Agent-Oriented Software Engineering II: Second International Workshop*, volume 2222 of *LNCS*, pages 17–31. Springer-Verlag.

[Pavón and Gómez-Sanz, 2003] Pavón, J. and Gómez-Sanz, J. (2003). Agent oriented software engineering with INGENIAS. In V. Marík and J. Müller and M. Pechoucek, editor, *Multi-Agent Systems and Applications III*, volume 2691 of *LNCS*, pages 394–403. 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2003), Springer Verlag.

[Penczek and Lomuscio, 2003] Penczek, W. and Lomuscio, A. (2003). Verifying epistemic properties of multi-agent systems via bounded model checking. In *Proc. of the second international joint conference on Autonomous agents and multiagent systems*, pages 209–216. ACM Press.

[Picard et al., 2002] Picard, G., Bernon, C., Gleizes, M., and Peyruqueou, S. (2002). ADELFE a methodology for adaptive multi-agent systems engineering. In Petta, P., Tolksdorf, R., and Zambonelli, F., editors, *Engineering Societies in the Agents World III: Third International Workshop (ESAW 2002)*, number 2577 in LNCS, pages 156–169. Springer Verlag.

[Rao, 1996] Rao, A. (1996). AgentSpeak(L): BDI agents speak out in a logical computable language. In van der Velde, W. and Perram, J., editors, *Agents Breaking Away: Proc. of the Seventh*

European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-96), volume 1038 of *LNAI*, pages 42–55. Springer-Verlag.

[Rao and Georgeff, 1991] Rao, A. and Georgeff, M. (1991). Modeling rational agents within a BDI-architecture. In Allen, J., Fikes, R., and Sandewall, E., editors, *Proc. of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484. Morgan Kaufmann publishers Inc.

[Ricordel and Demazeau, 2002] Ricordel, P. and Demazeau, Y. (2002). Volcano, a vowels-oriented multi-agent platform. In *From Theory to Practice in Multi-Agent Systems, Second International Workshop of Central and Eastern Europe on Multi-Agent Systems (CEEMAS 2001)*, volume 2296 of *Lecture Notes in Computer Science*, pages 253–262. Springer.

[Ricordel and Demazeau, 2000] Ricordel, P.-M. and Demazeau, Y. (2000). From analysis to deployment: A multi-agent platform survey. In *Working Notes of the First International Workshop on Engineering Societies in the Agents' World (ESAW-00)*, pages 93–105.

[Schreiber et al., 1994] Schreiber, A., Wielinga, J., Akkermans, J., and de Velde, W. V. (1994). CommonKADS: A comprehensive methodology for KBS development. Technical report, University of Amsterdam, Netherlands Energy Research Foundation ECN and Free University of Brussels.

[Shapiro et al., 2002] Shapiro, S., Lespérance, Y., and Levesque, H. J. (2002). The cognitive agents specification language and verification environment for multiagent systems. In *Proc. of the first international joint conference on Autonomous agents and multiagent systems*, pages 19–26. ACM Press.

[Shehory and Sturm, 2001] Shehory, O. and Sturm, A. (2001). Evaluation of modeling techniques for agent-based systems. In *Proc. of the fifth international conference on Autonomous Agents*. ACM Press.

[University of Waikato, 2004] University of Waikato (2004). *WEKA 3*. `http://www.cs.waikato.ac.nz/ml/weka/`.

[Wooldridge and Ciancarini, 2000] Wooldridge, M. and Ciancarini, P. (2000). Agent-Oriented Software Engineering: The State of the Art. In Ciancarini, P. and Wooldridge, M., editors, *First International Workshop on Agent-Oriented Software Engineering*, volume 1957, pages 1–28. Springer-Verlag, Berlin.

[Wooldridge et al., 2000] Wooldridge, M., Jennings, N., and Kinny, D. (2000). The Gaia methodology for agent-oriented analysis and design. *Journal of Autonomous Agents and Multi-Agent Systems*, 3:285–312.

[X.900, 1995] X.900 (1995). ISO/IEC IS 10746-x ITU-T Rec. X90x, ODP Reference Model Part x.

[Yu, 1997] Yu, E. K. (1997). Towards modelling and reasoning support for early-phase requirements engineering. In *Proc. of 3rd International Symposium on Requirements Engineering*, pages 226–235. IEEE.

[Zambonelli et al., 2002] Zambonelli, F., Bergenti, F., and Dimarzo, G. (2002). Methodologies and software engineering for agent systems. *AgentLink News*, 9:23–25.

[Zambonelli and Parunak, 2002] Zambonelli, F. and Parunak, H. V. D. (2002). Signs of a revolution in computer science and software engineering (esaw 2002). In *3rd International Workshop on Engineering Societies in the Agents' World*, volume LNCS 2577, pages 13–28. Springer Verlag.

[ZEUS, 1999] ZEUS (1999). `http://www.labs.bt.com/projects/agents/zeus/index.htm`.