An Overview of Current Trends in European AOSE Research

Carole Bernon
IRIT – University Paul Sabatier, 118 Route de Narbonne, 31062 Toulouse, France
E-mail: bernon@irit.fr, http://www.irit.fr/SMAC

Massimo Cossentino
ICAR-CNR, National Research Council, Viale delle Scienze, ed. 11, 90128 Palermo, Italy
E-mail: cossentino@pa.icar.cnr.it, http://www.pa.icar.cnr.it/~cossentino

Juan Pavón
Fac. Informática, Universidad Complutense Madrid, Ciudad Universitaria s/n, 28040 Madrid, Spain
E-mail: jpavon@sip.ucm.es, http://grasia.fdi.ucm.es/jpavon

Keywords: Agent Oriented Software Engineering (AOSE), Agent oriented methodologies, Multi-Agent Systems

Received: June 31, 2005

The agent oriented approach is doing great steps towards its (not yet reached) maturity; from a software engineering point of view, it is today positively used for the analysis and design of complex systems. In this paper, which is related to the activity of the AgentLink AOSE TFG (Agent Oriented Software Engineering Technical Forum Group), we provide a perspective of current research trends in this area with a specific attention for results coming from European groups. We start with a discussion of what are agents, specially from the perspective of the software engineer. We present recent trends in modelling agents and multi-agent systems, and then we review the different activities in the agent development process: from analysis and design to implementation, verification and finally testing.

Povzetek: Podan je povzetek evropskega raziskovanja AOSE.

1 Introduction

With the increasing amount of successful applications and techniques based on the agent paradigm, which have validated the feasibility of the approach, there is a big concern on its applicability in an industrial context. This implies the definition of repeatable, reusable, measurable and robust software process and techniques for the development of multi-agent systems (MAS). For this reason, a lot of effort in the agent field has been devoted to the definition of methods and tools for supporting agent oriented software engineering (AOSE). This involves the definition of modelling languages for the specification of MAS, techniques for requirements elicitation and analysis, architectures and methods for designing agents and their organizations, platforms for implementation and deployment of MAS, and validation and verification methods. Taking into account the diversity of influences in the agent paradigm (from distributed objects to knowledge base systems, but also from other fields such as Psychology, Biology and Social sciences) there are many methodological approaches, which should get unified and integrated in a common body of knowledge and practices. This is one of the aims of current actions at EU level, such as the AgentLink (www.agentlink.org) effort, or the collaboration in standardization organizations such as FIPA (www.fipa.org).

In this paper we try to provide a perspective of current research trends in this area, specially in EU groups. This can be useful as a starting reference point to look for specific matters (in this sense there is an extensive bibliography), and is complemented in relevant topics with other papers in this special issue.

The paper starts with a discussion of what are agents, specially from the perspective of the software engineer (section 2). This is followed by a presentation of trends in modelling this kind of systems (section 3). Then, different activities in the development process for MAS are reviewed: analysis and design (section 4), implementation (section 5), verification and testing (section 6). Finally, the conclusions (section 7) provide a view, from the authors of this paper, on what lines of work and trends should follow research in this area.

2 From Objects to Agents and Multi-Agent Systems

When dealing with the agent notion and how to engineer agent-based applications, one question often arises: may agents be considered as an extension of objects and then classical object-oriented software engineering be used as well to build agent-based applications? Several papers have tried to answer this question [76][106], others have compared agents with programs [46] or with components [7]. Many authors agree on the fact that distinguishing agents and objects is difficult because they share some aspects, but they also differ, mainly on notions such as autonomy and interaction. Both agents and objects encapsulate their state, which in objects is determined by the values of a set of variables whilst in agents this can be defined in terms of goals, beliefs, facts, etc., what determines a *mental state*. Objects may have control over their state by using private attributes or methods but any

public method of an object can be invoked by another object forcing the former one to perform the action described by the method. An object, contrary to an agent, has then a limited control over its behaviour because the decision on which method to execute is taken by an external actor (the *caller*). An agent can determine which behaviour to follow (depending on its goals, its internal state and its knowledge from the environment) and not because someone else forces it to do something. Therefore, the notion of autonomy is stronger in agents.

This autonomy in agents implies that usually they have their own thread of control, whilst, most of the time, objects are passive entities, becoming active just when one of their methods is invoked by another object. This difference may be alleviated by the notion of active objects in which an object has its own thread of control. However, agents have some features which make them something more than active objects. According to Van Parunak and Odell [76], agents exhibit a *dynamic autonomy* (their behaviour can be *reactive* as they react to changes in their environment, *proactive* as they are able to take initiatives to proceed into goal-directed actions, and *social* as they communicate with other agents in organizations) as well as an *unpredictable autonomy* (their behaviour depends on their state, their individual goals, and their interactions with others). Active objects would become agents if they are able to take "initiatives". However, this distinction is not always well established. For this reason some works in the agent domain, for instance, on formalization of coordination issues, usually are more related to classical concurrency theory and do not consider intentional aspects of agents.

What makes really the difference, according to many authors is the social dimension of agents (for instance, the Huhns-Singh test [58] states that *a system containing one or more reputed agents should change substantively if another of the reputed agents is added to the system)*. Agents cannot be considered in isolation and are social entities, which communicate and interact with other entities that share a common environment. Communication between objects is defined in terms of messages that activate methods, but in the agent domain, this communication is richer both in the diversity of mechanisms and in the language, which is defined at a more abstract level, in terms of ontologies and speech acts, for instance. This social perspective is reflected also in the definition of organizations with social rules and relationships among agents [42].

Therefore, the use of object-oriented software engineering techniques can be applied for the development of MAS, but some extensions are required to deal with social issues (organization, interaction, coordination, negotiation, cooperation), more complex behaviour (autonomy, mental state, goals, tasks), and a greater degree of concurrency and distribution.

2.1 Definition of Agents

In [91], an agent is defined as anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors. For Ferber, agents are still plunged into an environment but he endows agents with additional characteristics [41]. An agent becomes then able to communicate directly or not with other agents, it is driven by a set of tendencies, possesses resources of its own, has a limited representation of its environment, possesses skills and can offer services, and may be able to reproduce itself. Its behaviour tends towards satisfying its goals, taking into account the resources and skills available to it in accordance with its perception, its representation and the communication it receives. Depending on the nature of applications in which agents are used, different labels exist for agents [46][77]: agents are qualified as being autonomous, intelligent or mobile, for instance. This plethora of labels makes the term "agent" almost meaningless because it can be used too frequently to characterise anything, so [69] recommends to formally define the notion of agency. In this paper agents are characterized through their essential properties: an agent is able to act, is autonomous, proactive, communicates with others, and perceives its environment[1].

2.2 Definition of Multi-Agent Systems (MAS)

Most of the authors agree on viewing a MAS as a system composed of agents that communicate and collaborate to achieve specific personal or collective tasks. This is related to what was said before, an agent is not an isolated entity but it is only understandable when located in an environment where other agents exist, with which it can interact.

MAS are appropriate to deal with complex and open problems. The organization facilitates managing complexity by determining structures, norms and dependencies. In some cases, the organization is explicitly a subject of analysis and design (e.g., [42] [111]). But in certain approaches, the organization emerges at run time (e.g., [10][36][93]). This allows the analysis of emergent behaviours in systems in which is not easy to know their structure in advance. From the point of view of AOSE, this means that both top-down and bottom-up approaches are feasible when building a MAS, depending on the problem under study.

2.3 MAS Meta-models

Meta-modelling is a means to define concepts used in a system. This can facilitate analysis and design by identifying activities for instantiating the meta-model entities with respect to the target application (i.e., the meta-model identifies which elements should the developer look for, and what relationships and constraints exist for those elements). For instance, Aalaadin defines one of the first meta-models for MAS in terms of three main concepts: Agents, Groups and Roles [42]. With this meta-model, the developer has an organizational-driven approach to build a MAS. An organization is a structural relationship between a collection of agents and is described by a set of interaction modes. Agents are defined by their function in the organisation (Role) and belong to one or more Groups, possibly for gaining some capabilities.

[1] Properties defined during the second meeting of the AgentLink3 AOSE TFG (Ljubljana, February 2005).

Meta-models are also useful to integrate concepts. This is the approach of the MESSAGE project [21], whose aim was to define a methodology for the development of telecom applications using agent technology. MESSAGE adopted concepts and notations from different methodologies in a common framework. Its definition was made using meta-models. Furthermore, these meta-models were used to build graphical editors [51]. In order to cope with complexity of MAS, MESSAGE structured the specification of meta-models in five viewpoints: organization, agent, goals/tasks, domain, and interactions.

In the object world, the notion of object is clearly defined by a set of criteria and almost all developers agree on what makes a system object oriented. Meta-modelling is then possible relying on standard notations such as UML [90]. On the contrary, no universally accepted structural representation of the elements (agent, role, behaviour, ontology, etc.) that compose an actual MAS, with their relationships, exists yet. This has led several existing agent-oriented methodologies to propose their own concepts and system structure illustrated by a particular MAS meta-model. This lack of unification at the MAS meta-model level, and then at the agents concepts level, therefore prevents developers from reusing fragments of existing agent-oriented methodologies to build their own methodology especially dedicated to their needs (this is the methodology composition process suggested by the method engineering approach [17][54]; this proposes to create a new methodology starting from existing methodology parts, called method fragments, that a method engineer defines and stores in a method base).

A further step in this direction would be the standardisation of the process that is necessary to follow in order to build a new methodology. This would be desirable to make agent-oriented engineering used in the industrial world. From this perspective, some initial attempts have been made to find a unified meta-model based on several methodologies [11], or by trying to reach an agreement in the agent community with the work of the FIPA Modelling TC or the AgentLink III AOSE TFG.

3 Modelling Agents

Modelling agents and MAS needs adapted modelling languages, notations and tools. Agents, as said above, are not far from objects and most of the modelling methods are based on tools coming from the object-oriented domain. The most generally accepted modelling language used for object-oriented software engineering is UML. UML is a *de facto* standard, and most modelling tools are already based on it, which facilitates the development of tools. However, UML does not provide all the notation elements to model all the specific features of agents.

UML extension abilities (i.e., stereotypes, tagged values, constraints) have been used to support agent-oriented modelling. For instance, Agent-UML (AUML) [3] extends UML sequence diagrams to specify Agent Interaction Protocols by providing mechanisms to define agent roles, agent lifelines (interaction threads, which can split into two or more lifelines and merge at some subsequent points using connectors like AND, OR or XOR), 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, and whether messages are synchronous or not).

Also in the context of AUML there are proposals for extending class diagrams into agent class diagrams [2]. Here an agent class consists of several elements:

- An agent name used to differentiate objects from agents in a diagram and providing an agent with three information: instance, role and class.
- A state description that looks similar to the attribute compartment in class diagrams but expresses well-formed formulae for logical descriptions of the state, it may be used to model beliefs, desires and intentions of agents, for instance.
- Actions that can be reactive or proactive.
- Methods implementing services, as in UML classes.
- Capabilities describing what an agent can do.
- Organisation belonging, which specifies the different groups in which an agent evolves, the roles it plays and under which constraints it evolves in these groups.
- Agent head automata, which define the behaviour of an agent.

AUML is under study in the FIPA Modelling TC, and being modified in order to take into account new features in UML 2.0 [57]. For example, communication between agents are now captured by enhanced sequence diagrams, which become interaction diagrams, in which agents can change their role, add or delete roles during interactions, and notions of loop or break are added to the AND, OR and XOR connectors that were available. AUML is being smoothly introduced as an add-on into different agent-oriented toolkits, such as OpenTool for ADELFE [10] and the INGENIAS Development Kit [82].

Another proposal for agent-oriented modelling as an UML profile is AORML (Agent-Object-Relationship Modelling Language) [104]. Here, agents are considered from two perspectives: external and internal. The external AOR model describes the perspective of an external observer who is watching the agents and their interactions. Agent Diagrams are used to depict the agent types and objects of the domain and their relationships, while interactions are modelled using Interaction Frame Diagrams (possible interactions between two agent types), Interaction Sequence Diagrams (instances of interaction processes) and Interaction Pattern Diagrams (general interaction patterns). An internal model adopts the view of a particular agent to be modelled and depicts, using three kinds of diagrams (Reaction Frame Diagrams, Reaction Sequence Diagrams, Reaction Pattern Diagrams), the world represented by the mental state of this agent.

A more recent extension of UML for MAS is AML, which is described in another paper of this special issue [25]. Two UML profiles for AML are given and enable implementing AML in CASE tools based on UML 1.* or UML 2.0. Furthermore, using these AML profiles, a

designer is free to customise AML through the definition of extensions to this language.

There are also approaches based on OPM (Object Process Methodology) [37]. OPM considers processes and objects as equally important classes of things, which together describe the function, structure and behaviour of systems. A single diagramming tool, Object-Process Diagrams (OPDs), is enough for modelling the system. This has been extended in OPM/MAS [99] by taking MAS building blocks from Gaia methodology. For instance, organization, society, platform, rule, role, user, protocol, belief, desire, fact, goal, intention, and service are modelled as OPM objects. And the behavioural concepts such as agent, task, and messaging are modelled using the process concept. Another approach [74], taking inspiration from OPM, allows zooming through different abstraction layers and apply this feature to SODA [79], a methodology that addresses the coordination aspects of agent societies. The analysis of complexity is also considered from the perspective of interactions in [83].

In section 4.3 we discuss how the management of complexity can be also addressed by considering complementary aspects of a MAS.

4 Analysing and Designing Agents

According to Sommerville [97], all the different kinds of software development processes share some fundamental activities. These include specification (consisting in the definition of software functionalities and constraints, i.e., requirements analysis), design and implementation (consisting in the production of the software), validation (where the produced software should be validated against customer requirements) and evolution (the software evolves according to customer new needs). In this section we discuss the first two items of this list: specification analysis and design.

During the specification phase, the designer collects and analyzes the software requirements, which are usually considered from two perspectives: User and System. The latter being the detailed and more technical expression of what the customer specifies in the User Requirements. System Requirements consist of functional (services the software should provide), non–functional (constraints on the services) and domain requirements (coming from the application domain).

Design consists in converting system specifications into an executable system. This is usually achieved by structuring the software into modules, defining the data to be managed and the interfaces between components. Sometimes a specific attention is given to the algorithms that are necessary to solve the problem.

A fundamental contribution in defining the impact of agents in these phases has been argued by Jennings [63] in the sense that agents can be a successful solution for two major problems of contemporary approaches: rigidity of components interactions and limitedness of available system's organizational structures.

In the next sub-sections we present existing contributions in this area (with a specific attention for European ones) according to their key features. Specifically, we consider formal and non formal approaches, multi-views paradigms, agent design life cycles and some other remaining issues.

4.1 Formal approaches

Many authors looked at the problem of analysis and design of agent-oriented systems with a formal approach. This usually includes the adoption of a mathematical formalism to obtain a correct specification of the system to be; the output of a formal method is a formal specification that can be used for implementing the system, verifying its correspondence with user requirements or evaluating the final result [85].

Several of these works adopt a kind of logic (usually a modal logic [96]) to represent the system. As an example, LORA (Logic of Rational Agents) [107] which is founded on a first-order logic, includes a BDI (Belief-Desire-Intention) [87] component (used for the agent architecture), a temporal component (used for specifying the system dynamics), and an action component (used to represent agents' actions). LORA is adopted by MABLE (a language for the design of MAS) that allows an automatic verification of the agent system [108].

Situation Calculus [71] is another expression of this field of research; it is a first-order logic (with some extensions to second-order logic) capable of representing dynamic domains. IndiGolog [34] is a recent implementation of situation calculus, supporting the high-level programming of robotic intelligent agents that can perform online planning and plan execution in partially unknown environments. In IndiGolog (that is part of the GOLOG [67] family), environment dynamics is modelled using situation calculus while the agent behaviour is designed in a procedural way.

Another formal approach is due to M. Luck and M. D'Inverno [69] and it is an application of the Z language [98] to the specification of agents. Z is based on first order predicate calculus with the original introduction of the concept of schema. A schema is composed of a declarative part (declaration of variables and their types) and another part where variables are related and their constraints expressed. Agents in Z are defined within a four-layer hierarchy that includes: entities (inanimate objects with attributes), objects (entities with capabilities), agents (objects with goals), autonomous agents (agents with motivations). In this work the authors take profit of the great number of existing experiences in Z for inheriting a great number of tools that include code production and model checking capabilities. Another approach that uses the Z formalism (and statecharts) can be found in [56].

4.2 Non-formal Approaches

Non-formal approaches to the specification and design of agent systems are mostly based on the use of structured natural language and graphical notations. Among these, for system requirements specification, UML-related diagrams like use-case and sequence diagrams are very common use. These approaches are mainly requirement-oriented and they often aim at capturing system functionalities through a set of heuristics and views.

Several agent-oriented design methodologies perform the specification in this way; they generally

include a complete design process, not only system specification aspects. We can fundamentally identify three categories of non-formal specifications: functional-oriented [62] (often adopting use-case diagrams), goal-oriented approaches [103] (that aim at identifying the goals of the system and eventually dividing them among agents), and, finally, role-oriented approaches [65] (they adopt the role as the key abstraction for specifying a MAS, they are often also concerned about designing roles/agents coordination). While the functional and goal-oriented specifications are well-known and widely adopted in the object-oriented context, role-guided specifications are more specific of the agent community.

Functional specifications (mostly looking at European works) are adopted in the PASSI methodology [29] and the ROADMAP [64], an extension of Gaia [109], both of them adopting use-case diagrams.

PASSI starts analysis with use-cases and arrives to code production and testing in an iterative process. It includes an extensive patterns reuse practice and it is conceived to be supported by a specific design tool (PTK), since several of its activities are partially automated.

Identification and modelling of system goals is part of the MESSAGE methodology [21], which is based on a set of meta-models supporting five different views of the MAS: organization, agent, tasks/goals, interactions, and environment. INGENIAS [82] refines and extends these meta-models, and uses them to build support tools for all stages of the development cycle. Furthermore, for requirements elicitation, INGENIAS proposes to base on Activity Theory to analyse intentional and social issues of the system, by providing a set of contradiction patterns that guide the developer in the identification of conflicts in the specification about the agent and the organization goals [47].

Tropos [16] starts from the i* framework [110], which has been developed mainly thinking on information systems, actors, beliefs, commitments and goals are used to model system organization. Tropos uses this requirements analysis approach and incorporates it in a complete process that moves from the specification to detailed design.

One of the key features of agency consists in interaction; we can even note that this is also the fundamental aspect of some standardization attempts coming from FIPA (Abstract Architecture Specification [43]) or OMG MAF (Mobile Agent Facility [78]). As a consequence, many authors devoted their attention to capturing interaction aspects often by modelling agents' roles [65][18].

European methodologies that give a prominent importance to role modelling are Gaia [109], SODA [79] and RICA [94] (but also the cited MESSAGE, INGENIAS, and PASSI).

Gaia has been, probably, the most influent methodology concerning the analysis of the system as a society/organization consisting on a set of roles that are later assigned to agents. Gaia's roles are related with one another, and participate in pre-defined patterns of interactions with other roles. Implementation issues are not dealt in this methodology since considered depending on the chosen deployment agent platform. Although initially Gaia suffered from the limitation of being conceived for closed systems and ignoring the possibility of self-interesting agents, a new release of it [111] included concepts like organizational rules as the way to manage more complex open systems.

SODA (Societies in Open and Distributed Agent Spaces) [79] aims at modelling the behaviour of agent societies (considered as not deducible from the behaviour of single agents) and their environments (that can be open, distributed, dynamic and unpredictable). It has a specific attention for agent interactions (starting from a role model) but does not face the design of the agent's inner structure.

Another methodology that puts in a prominent position roles is RICA (Role/Interaction/Communicative Action) [94]. It integrates relevant aspects of Agent Communication Languages (ACL) and Organisational Models and it is itself based on the concepts of Communicative Roles and Interactions.

Other authors concentrated their efforts to coordination among agents [27][80]. A coordination-based approach should consider system openness, the presence of self-interested agents and MAS social laws that rule the overall behaviour of the agents thus encompassing single-role modelling issues.

Coordination is sometimes pursued by adopting a programmable coordination media (like the MARS system presented in [19]), but other authors specifically conceived their design methodologies for dealing with coordination.

Another interesting methodology specifically conceived for coordination of robotic agents is Cassiopeia [38]. Cassiopeia design process is based on the concept of role, agent, dependency, and group; an agent is seen as a set of roles (there can be individual roles, relational roles and organizational roles). The methodology enumerates several different layers, among them the organizational roles layer describes the dynamics of the groups by defining the roles that the agents have to play to let the group appear. Dependencies among roles can be of three types: functional, resource-based or goal-based and in this sense the methodology partially recalls the already cited i* framework.

Cassiopeia assumes that agents are cooperative and this is the same hypothesis that is behind the ADELFE methodology, which focuses on adaptive MAS [9]. Adaptive software can be profitably used in situations in which the environment is unpredictable or the system is open. Contrary to Cassiopeia, in ADELFE agents are not characterised by roles but by the cooperation rules they follow. These rules are described in a proscriptive way, they express what are non cooperative situations, and make an agent locally decide why and when changing its interactions with others. Cooperation is thus viewed as the engine of adaptation according to the AMAS (Adaptive Multi-Agent System) theory [22].

Other contributions about non-formal agent design come from MaCMAS/UML [84], which is a fragment of methodology devoted to deal with large/complex MAS, and the works on modelling electronic institutions and their norms in Islander [95].

4.3 Multi-view Approaches

Multi-views, multi-perspectives, multi-level approaches base their philosophy on three well-known methods for tackling complexity, already mentioned by Booch [12]: Abstraction, Decomposition, Hierarchy. After all, as it can be deduced from the discussion in sections 2 and 3, agent-oriented systems can be more complex than object-oriented ones and therefore a well structured way to manage this complexity is necessary.

The structuring of a MAS in several viewpoints appears in many methodologies. One of the first to propose this was Vowels Engineering, which has been the basis for the MAGMA approach [35]. It considers the five Latin vowels (initially only the first four): Agent, Environment, Interactions, Organization, and User. Different techniques can be applied to analyse 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. These viewpoints have been applied similarly in the MESSAGE [21] and INGENIAS methodologies [81], which redefine viewpoints as organization, agent, domain/environment, goals/tasks, and interactions.

The concept of level in agency is also another way of considering several views. It has been initially introduced by Newell [75] and Jennings [63] recalled the knowledge level and complemented it with a new social level. The knowledge level is concerned with the agent seen as an asocial problem solver while the social level looks at the agent organization as its main focus.

Other works in this direction presented different *perspectives* [28][32], which are more directed to the representation of the system from a different point of view (architectural, social, knowledge, computer, resource, autonomy) rather than a different level of abstraction.

Other examples of methodologies that emphasize the modelling of the MAS from different viewpoints are MAS-CommonKADS [60] (organization, tasks, experience, agents, communications, coordination, and design), ODAC [50], which uses the five ODP viewpoints (enterprise, information, computational, technology and engineering) [61], and MASSIVE [68] (that includes seven views: environment, task, role, interaction, society, architectural, system).

4.4 Agent Design Life Cycle Models

The whole set of activities and phases needed to develop and maintain a software system is usually addressed as a *Software (Engineering or Development) Process*. Fuggetta in [48] defines it as "the coherent set of policies, organizational structures, technologies, procedures, and artifacts that are needed to conceive, develop, deploy, and maintain (evolve) a software product", sometimes this is also known as a *Software Life Cycle Process* [59]. Usually the sequence of phases (here we mean high level activities or set of activities) that compose a Software Process is ruled by a software life cycle model. Examples of software life cycle models are the waterfall model, the prototyping model, the evolutionary development, the incremental/iterative delivery, the spiral model, and so on.

A classification of many agent-oriented methodologies according to the software life cycle model they adopt, can be found in [24]. The paper remarks that current research in the area of AOSE methodologies underestimate the importance of the process model in the development of MAS; according to the authors, this is confirmed by the fact that in many cases, AOSE methodologies do not make explicit reference to the underlying process model. Anyway, most of them propose iterative and incremental development process in the same way as the Unified Process.

Some novelties about life cycle models for agents come from the application of the Extreme Programming [5] and Agile Manifesto [4] principles to agents. Proposed design approaches [26][66] seem to show that besides the respect for the main principles of this research stream (attention for code rather than documentation, central role of customer, and so on) a fundamental importance in MAS agile design is played by its ontological aspects (both of the cited approaches give great importance to drawing some ontological models of the problem domain).

4.5 Other Issues In Designing Agents

Despite of the number of works we have discussed, we are still leaving apart some specific areas. These for instance include the design of Internet specific applications by means of agents (see [112]); the importance of this field is growing up in conjunction with the studies on web-services [72] (and their extensions to agent-services [33]).

Another important aspect of design is evaluation. In the last years several works have been proposed on this topic. Some look at specific attributes of the methodology to evaluate it (this is the case of [23][100]) while some others more generically try to identify the elements that a methodology should include to deal with specific aspects of agency like for instance managing complexity [83].

Finally, we would like to report some studies on the composition of new methodologies based on the reuse of existing portions of them (usually called *method fragments*). These works start with experiences from classical software engineering [17][86] and have their primary justification in the claim that one single design methodology cannot be suited to face all problems and developing contexts. According to this paradigm, each class of problems should be faced by a specific methodology that properly considers the skills of the

development group and other factors affecting the software production (like for instance strategic choices about implementing environment and technologies).

Actually, a wide repository of method fragments coming from diffused agent methodologies (Gaia, MaSE, PASSI, Prometheus, Tropos) is included in the Open Process Framework [44]. A similar approach is pursued by the FIPA Methodology Technical Committee, whose results can also be found in works of some of its members [31][45]. Although some experiences exist in supporting tools for object-oriented approaches [102], the lack of specific agent-oriented instruments and the intrinsic complexity of the approach has still limited the diffusion of this paradigm.

5 Implementing Agents

Agent systems can be implemented and deployed on a variety of target platforms. There are agent-oriented platforms that conform to some standards such as FIPA or MAF [78], but it can be the case that a MAS is finally realized on more conventional technology, for instance, as Java distributed objects or components. Here we describe both agent platforms (section 5.1) and proposals for transformation from MAS design models to implementation (section 5.2). Finally, in section 5.3 we consider agent-oriented programming languages.

5.1 Agent Platforms

Agent platforms support developers by providing a set of reusable components and services for the implementation and deployment of agents. Most of them are compliant with standards. In Europe, JADE can be considered as the reference FIPA compliant platform. Other platforms are more focused to support agent coordination, such as TuCSoN and Islander.

JADE (Java Agent DEvelopment Framework) [6] originates as a collaboration between the research labs of Telecom Italia (TILAB) and Univ. Parma, and currently is distributed as open source software under the terms of the LGPL (Lesser General Public License Version 2). JADE illustrates well the implementation of FIPA management architecture components: the Agent Communication Channel, the Agent Management System, and the Directory Facilitator. Agent communication is performed through message passing, where FIPA ACL is the language to represent messages, and with libraries that implement FIPA protocols, which can be used as reusable components when building agent-based applications. This facilitates the task of developers who can rely on agent lifecycle management by JADE and have some guarantee of interoperability with other FIPA compliant agent systems. JADE supports both reactive and deliberative agents by defining a structure for agent behaviours, which can be Java classes implementing state machines or rule systems, by an integration of JESS (Java Expert System Shell, available at http://herzberg.ca.sandia.gov/jess/) in the platform. Furthermore, JADE provides some tools for agent debugging (sniffer agents) and monitoring, and other common services such as naming and yellow pages. As a result of the EU IST project LEAP (Lightweight Extensible Agent Platform), JADE incorporated facilities for agent mobility and can be deployed on mobile lightweight Java environments down to J2ME-CLDC. Currently, LEAP libraries are distributed as an add-on of JADE distribution from version 3.0 onwards. A board has been constituted recently with the purpose of driving its evolution and consolidating JADE as a *de-facto* standard middleware for agent-based applications.

Another approach for agent communication, instead of message passing, is the use of a tuple spaces, a classic mechanism for coordination. This is illustrated by TuCSoN (Tuple Centres Spread over the Networks), by the Univ. Bologna [88]. An interesting feature of this kind of systems is the ability to define coordination laws (something that is not common for tuple space approaches in general). Islander+AMELI [40] also provides a coordination middleware, by exploiting the concept of electronic institutions to implement complex negotiation processes.

5.2 Transformation from Design to Implementation

As a modelling paradigm agents contribute to the use of abstract concepts that are close to those used when reasoning about human behaviours and organizations. This can facilitate analysis and design activities but the gap to implementation is greater than with other paradigms, which are closer to current computational frameworks. In this sense, although there are well-established agent platforms, such as JADE, it is common to see agent systems that are implemented on more conventional platforms, usually depending on the application environment and constraints (for instance, a robotic system or a J2EE server). In order to solve this kind of situations, some integrated development environments (IDEs) provide tools for modelling with agent concepts and a process for transforming agent specifications into code for the target platforms.

Finally, when considering multiple target platforms, the trend is to follow the OMG Model Driven Architecture (MDA) approach [73]. Basically, the idea is to specify the meta-model of a MAS modelling language, which is platform independent, and those of the target platforms. Mappings define rules or algorithms that determine how instances of types in the MAS meta-model result in the generation of instances of types in the meta-model specifying a target platform. This approach has been discussed in [1] and is used by the INGENIAS Development Kit (IDK) to generate code on JADE, Servlets, Robocode tanks, and other systems [51]. It is also proposed by MetaDIMA [52] and Agent Factory [30].

5.3 Agent-Oriented Programming Languages

The use of agent-oriented programming languages facilitates the understanding of agent features. There is an extensive review of this in an accompanying paper of this special issue [13], which considers imperative, declarative and hybrid approaches. Basically, the different proposals consider an agent model that makes

emphasis either on mobility issues, or on an intentional behaviour model, or on a communication model. CLAIM [39] is probably the most complete in considering all these issues and being applied to real applications. Many provide support for a BDI model, such as dMARS, 3APL, or Coo-BDI.

6 Verification and Testing

Verification and testing techniques for MAS usually apply known results from concurrent and distributed computing.

Verification is normally based on formal theories, that allow the analysis of a system in order to determine whether certain properties hold. These can be *liveness* (whether the system will progress) or *safety* properties (whether the system will do right things), thus answering to the question *is the system being built right?* When the property consists on whether the application fulfils the requirements, we usually refer to it as *validation*. Testing, on the other hand, is usually defined as the activity of looking for errors in the final implementation.

What is interesting to note in the case of MAS when discussing verification and testing is whether organizational, cognitive, development, evolution, and motivational concepts are considered, because the consequences of having concurrent and distributed processes are already a subject extensively covered in the literature since the seventies. Winograd & Flores [105] already criticised that many approaches try to work with these properties through techniques that were conceived for other purposes, without taking advantage of specific agent characteristics. In this context, verification and testing of MAS have not just imported techniques from other paradigms, but they have also created new approaches to solve this problem.

An example of the first formal approaches for verification in the agent domain is DESIRE [15], a design and specification framework that describes agents and the MAS itself as networks of tasks organized in a hierarchy. The interaction and coordination among agents is specified as interchanges of pieces of information and control dependencies. Properties to be verified are represented with temporal logics: what is a conflict among goals or how to choose among design alternatives. Checking properties consists of demonstrating that these are satisfied in a concrete problem using the DESIRE representation of the system. Although this allows proving complex properties of the system and the domain, it has the limitation of the agent model as being task-oriented.

Other formal approaches have shown limited scope because they are assuming a fixed agent model, usually more as a kind of reactive process rather than intentional, and demand too detailed specifications, which makes these techniques work for toy examples but unaffordable for real cases, apart of the learning curve that they imply for developers. For these reasons, there are several approaches that try to mix the goodness of formal languages with the expressive power of semi-formal (usually graphical) languages.

An example of this is the use of model checking techniques to verify the satisfaction of requirements in Tropos [49]. Specifications with the graphical language of Tropos are translated into Formal Tropos, adding temporal logic constructs. This offers the possibility of verifying the specification with formal methods.

Recently we start to see the application of theories coming from other fields, such as Sociology. Activity Theory, for instance, has been applied to the identification of contradiction patterns (e.g., conflicts between individual goals and community goals) by translating concepts for the social science to agent concepts, in this case for INGENIAS and Tropos [47]. Activity Theory is also being considered for analysing social coordination in the TuCSoN platform [89].

Concerning testing, apart of debugging tools that help the developer to follow messages exchange and in some cases to introspect agents (as in the case of MadKit [53]) an interesting approach is the use of data mining tools for analysing and presenting results to the developer. This is used for the JADE platform in the ACLAnalyser tool [14]. Another work, specifically conceived for the JADE platform and including both a test method (aimed at testing single agent features with a regression testing approach) and a supporting tool is presented in [20].

7 Conclusion

The agent-oriented approach, from a software engineering point of view, is mainly used for analysis and design of complex systems. Implementation and deployment of these systems may take a variety of forms, sometimes following agent related standards (such as FIPA and MAF) but usually as conventional distributed objects or component based software. Thus, the main benefit of agent-orientation at present seems to be at the level of modelling. The coupling with that diversity of target platforms is motivating approaches in the AOSE community which are in line with the OMG Model Driven Architecture (MDA) approach. Following this, and considering the state-of-the-art as reported in this work, we think the agent approach can be profitably used for modelling the solution at a platform independent level, and then some tools could provide proper transformations to specific target platforms.

Acknowledgement

We would like to thank all the members of the AgentLink AOSE Technical Forum Group for their active participation during the AL3 Technical Fora and their contribution in the off-line work.

References

[1] Amor M., Fuentes L. and Vallecillo A. (2005). Bridging the Gap Between Agent-Oriented Design and Implementation Using MDA. In: *Agent-Oriented Software Engineering V: 5th International Workshop, AOSE 2004*. Lecture Notes in Computer Science 3382, Springer Verlag, pp. 93—108.

[2] Bauer, B. (2002). UML Class Diagrams Revisited in the Context of Agent-Based Systems. In: *Agent-Oriented Software Engineering II: Second International Workshop*, *AOSE 2001*. Lecture Notes

in Computer Science 2222, Springer-Verlag, pp. 101—118.

[3] Bauer B, Müller J., and Odell J. (2001). Agent UML: A Formalism for Specifying Multiagent Interaction. In: *Agent-Oriented Software Engineering: First International Workshop, AOSE 2000*. Lecture Notes in Computer Science 1957, Springer-Verlag, pp. 91—103.

[4] Beck K., et al. *Manifesto for Agile Software Development*. http://www.agilemanifesto.org.

[5] Beck K., and Andres C. (2004). *Extreme Programming Explained: Embrace Change , 2nd Edition*. Addison-Wesley.

[6] Bellifemine F., Poggi A., and Rimassa, G. (2001). Developing multi-agent systems with a FIPA-compliant agent framework. *Software Practice and Experience 31* (2), pp. 103—128.

[7] Bergenti F., and Huhns M. (2004). On the Use of Agents as Components of Software Systems, In: [8], chapter 2, pp. 19—32.

[8] Bergenti F., Gleizes M.-P., and Zambonelli F., editors (2004). *Methodologies and Software Engineering for Agent System: The Agent Oriented Software Engineering Handbook*. Kluwer Academic Publisher, New York.

[9] Bernon C., Camps V., Gleizes M.-P., and Picard G. (2005). Engineering Adaptive Multi-Agent Systems: The ADELFE Methodology. In: [55], chapter VII, pp. 172—202.

[10] Bernon C., Camps V., Gleizes M.-P. and Picard G. (2004). Tools for Self-Organizing Applications Engineering. In: *Engineering Self-Organising Systems, Nature-Inspired Approaches to Software Engineering [revised and extended papers presented at the Engineering Self-Organising Applications Workshop, ESOA 2003].* Lecture Notes in Artificial Intelligence 2977, Springer Verlag, pp. 283—298.

[11] Bernon C., Cossentino M., Gleizes M-P., Turci P., and Zambonelli F. (2005). A Study of some Multi-agent Meta-models. In: *Agent-Oriented Software Engineering V: 5th International Workshop, AOSE 2004*. Lecture Notes in Computer Science 3382, Springer Verlag, pp. 62—77.

[12] Booch, G. (1994). *Object-Oriented Analysis and Design with Applications*. Addison-Wesley, Reading, MA.

[13] Bordini, R., Braubach, L., El Fallah-Seghrouchni, A., Dastani, M., Gomez-Sanz, J., Leite, J., O'Hare, G., Pokahr, A., and Ricci, A. (2005). A Survey on Languages and Platforms for MAS Implementation. *Informatica 29* (this issue).

[14] Botía J., López-Acosta A., and Gómez-Skarmeta A. (2004). ACLAnalyser: A Tool for Debugging Multi-Agent Systems. *Proc. 16th European Conference on Artificial Intelligence, ECAI 2004*, pp. 967—968.

[15] Brazier, F. M. T., Dunin-Keplicz, B. M., Jennings, N. R., and Treur, J. (1997). DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework. *International Journal of Cooperative Information Systems 6(1)*. pp. 67—94.

[16] Bresciani P., Giorgini P., Giunchiglia F., Mylopoulos J., and Perini A. (2004). TROPOS: An Agent-Oriented Software Development Methodology. *Journal of Autonomous Agents and Multi-Agent Systems*, Kluwer Academic Publishers 8(3), pp. 203—236.

[17] Brinkkemper S., Lyytinen K., and Welke R. (1996). *Method Engineering: Principles of Method Construction and Tool Support*. Chapman &Hall.

[18] Cabri G., Ferrari L., and Zambonelli F. (2004). Role-based Approaches for Engineering Interactions in Large-Scale Multiagent Systems. In: *Post-Proceedings of Advances in Software Engineering for Large-Scale Multiagent Systems (SELMAS 03)*, Lecture Notes in Computer Science 2940, Springer-Verlag, pp. 243—263.

[19] Cabri G., Leonardi L., and Zambonelli F. (2003). Engineering Mobile Agent Applications via Context-Dependent Coordination. *IEEE Transactions on Software Engineering* 28(11), pp. 1039-1055.

[20] Caire G., Cossentino M., Negri A., Poggi A, and Turci P. (2004). Multi-agent Systems Implementation and Testing, In: *From Agent Theory to Agent Implementation - Fourth International Symposium (AT2AI-4)*, Vienna, Austria.

[21] Caire G., Evans R. Massonet P., Coulier W., Garijo F.J., Gomez J., Pavón J., Leal F., Chainho P., Kearney P.E., and Stark J. (2002). Agent Oriented Analysis using MESSAGE/UML. In: *The Second International Workshop on Agent-Oriented Software Engineering (AOSE 2001)*, Lecture Notes in Computer Science 2222, Springer-Verlag, pp. 119-135.

[22] Capera D., Georgé J.P., Gleizes M.P., and Glize P, (2003). The AMAS Theory for Complex Problem Solving based on Self-organizing Cooperative Agents. In: *Proc. of the 1st International Workshop on Theory And Practice of Open Computational Systems (TAPOCS03@WETICE'03)*, Linz, Austria, pp.283—288.

[23] Cernuzzi L., and Rossi G. (2002). On the Evaluation of Agent Oriented Methodologies. In: *Proc. of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies*, pp. 21-30.

[24] Cernuzzi L., Cossentino M., and Zambonelli F. (2005). Process Models for Agent-based Development. *Engineering Applications of Artificial Intelligence* 18(2), pp. 205-222.

[25] Trencansky I., Cervenka R.. (2005). Agent Modeling Language (AML): A Comprehensive Approach to Modeling MAS, *Informatica 29* (this issue).

[26] Chella A., Cossentino M., Sabatucci L., and Seidita V. (2004). From PASSI to Agile PASSI: Tailoring a Design Process to Meet New Needs. In: *2004 IEEE/WIC/ACM International Joint Conference on Intelligent Agent Technology (IAT'04)*, Beijing, China. pp. 471-474.

[27] Ciancarini P. (1996). Coordination Model and Languages as Software Integrators, *ACM Computing Surveys*, 28(2), pp. 300-302.

[28] Cossentino M. (2002). Different Perspectives in Designing Multi-agent Systems, *AGES'02 workshop at NODe02*, Erfurt, Germany. pp. 61-73.

[29] Cossentino M. (2005). From Requirements to Code with the PASSI Methodology. In: [55], chapter IV, pp. 79—106.

[30] Cossentino M., Sabatucci L., and Chella A. (2003). A Possible Approach to the Development of Robotic Multi-Agent Systems. In: *IEEE/WIC International Conference on Intelligent Agent Technology (IAT'03)*, pp. 13-17.

[31] Cossentino M., and Seidita V. (2004). Composition of a New Process to Meet Agile Needs Using Method Engineering, In: *Software Engineering for Large Multi-Agent Systems vol. III*, Lecture Notes in Computer Science 3390, Springer-Verlag, pp. 36-51.

[32] Cossentino M., and Zambonelli F. (2004). Agent Design from the Autonomy Perspective. In: *Agents and Computational Autonomy: Potential, Risks, and Solutions*, Lecture Notes in Computer Science 2969, Springer-Verlag, pp. 140-150.

[33] Dale J., and Ceccaroni L. (2002). Pizza and a Movie: A Case Study in Advanced Web Services. In: *Agentcities: Challenges in Open Agent Environments Workshop*, AAMAS Conference 2002, Bologna, Italy.

[34] De Giacomo G., Lespérance Y., Levesque H.J., and Sardina, S. (2004). On the Semantics of Deliberation in IndiGolog - From Theory to Implementation. *Annals of Mathematics and Artificial Intelligence* 41(2-4), pp. 259-299.

[35] Demazeau Y. (1995). From Cognitive Interactions to Collective Behaviour in Agent-Based Systems, *1st European Conference on Cognitive Science*, Saint-Malo, France, pp. 117-132.

[36] Di Marzo Serugendo, G., Gleizes, M.-P., Karageorgos, A. (2005). Self-Organisation and Emergence in MAS: An Overview. *Informatica 29* (this issue).

[37] Dori, D. (2002). Object-Process Methodology: A Holistic System Paradigm. Springer.

[38] Drogoul A., and Collinot A. (1998). Applying an Agent-Oriented Methodology to the Design of Artificial Organisations: a Case Study in Robotic Soccer. *Journal of Autonomous Agents and Multi-Agent Systems*, 1(1), pp. 113-129.

[39] El Fallah Seghrouchni A. and Sun A. (2003). Claim: A Computational Language for Autonomous, Intelligent and Mobile Agents. In: *Proceedings of ProMAS'03,* Lecture Notes in Artificial Intelligence 3067, Springer Verlag, pp. 90–110.

[40] Esteva M., Rosell B., Rodríguez-Aguilar J.A., and Arcos, J.L. (2004). AMELI: An Agent-based Middleware for Electronic Institutions. In: *Third International Joint Conference on Autonomous Agents and Multi-agent Systems (AAMAS'04).* pp. 236—243.

[41] Ferber J. (1999). *Multi-Agent Systems*, Addison-Wesley: Reading, MA.

[42] Ferber J., and Gutknecht O. (1998). A Meta-model for the Analysis and Design of Organizations in Multi-agent Systems. In Proc. of the *3rd International Conference on Multi-Agent Systems (ICMAS'98)*, pp. 128–135.

[43] FIPA. Abstract Architecture Specification. Document SC00001L. Available online at http://www.fipa.org/specs/fipa00001/SC00001L.html.

[44] Firesmith D.G., and Henderson-Sellers B. (2002). *The OPEN Process Framework*. Addison-Wesley.

[45] Fortino G., Garro A., and Russo W. (2004). From Modeling to Simulation of Multi-Agent Systems: an Integrated Approach and a Case Study. In: Proceedings of the *Second German Conference on Multiagent System Technologies (MATES'04),* Lecture Notes in Artificial Intelligence 3187, Springer-Verlag, pp. 213-227.

[46] Franklin S, and Graesser A. (1996) Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents. In: *Intelligent Agents III – Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Lecture Notes in Artificial Intelligence, 1193, Springer Verlag, pp. 21—35.

[47] Fuentes R., Gómez-Sanz J.J., and Pavón, J. (2004). Social Analysis of Multi-Agent Systems with Activity Theory. In: *Proceedings of CAEPIA 2003*, Lecture Notes in Artificial Intelligence 3040, Springer-Verlag, pp. 526-535.

[48] Fuggetta A. (2000). Software Process: a Roadmap. In Proceedings of the *Conference on the Future of Software Engineering*, ACM Press, New York (USA), pp. 25-34

[49] Fuxman A., Pistore M., Mylopoulos J. and Traverso P. (2001). Model Checking Early Requirements Specifications in Tropos. In: *Proceedings 5th IEEE International Symposium on Requirements Engineering (RE 2001)*, pp. 174-181.

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

[51] Gomez-Sanz J. J., and Pavón, J. (2002). Meta-modelling in Agent-Oriented Software Engineering. In: *Advances in Artificial Intelligence - IBERAMIA 2002*, Lecture Notes in Artificial Intelligence 2527, Springer-Verlag, 606-615.

[52] Guessoum Z., and Jarraya, T. (2005). Meta-Models & Model-Driven Architectures, *Contribution to the AOSE TFG AgentLink3 meeting*, Ljubljana, 2005.

[53] Gutknecht O., Ferber J., and Michel F. (2001). Integrating Tools and Infrastructures for Generic Multi-agent Systems. In: *Proceedings of the fifth international conference on Autonomous agents (Agents 2001)*, ACM Press, pp. 441–448.

[54] Henderson-Sellers, B., and Debenham, J. (2003). Towards Open Methodological Support for Agent Oriented Systems Development. In: *Proceedings of the First International Conference on Agent-Based Technologies and Systems*. University of Canada, Canada. pp. 14–24.

[55] Henderson-Sellers, B. and Giorgini, P., editors (2005). *Agent-Oriented Methodologies*. Idea Group Publishing.

[56] Hilaire V., Koukam A., Grue, P., and Muller J.-P. (2000). Formal Specification and Prototyping of Multi-agent Systems. In: *Engineering Societies in the Agents' World (ESAW'00)*, Lecture Notes in Artificial Intelligence 1972, Springer Verlag, pp. 114—127.

[57] Huget M.-P., and Odell J. (2005). A Study of some Multi-agent Meta-models. In: *Agent-Oriented Software Engineering V: 5th International Workshop, AOSE 2004*. Lecture Notes in Computer Science 3382, Springer Verlag, pp. 16—30.

[58] Hunhns M., and Singh M.P. (1999). A Multiagent Treatment of Agenthood. *Applied Artificial Intelligence: An International Journal* 13(1-2), pp. 3-10.

[59] IEEE Computer Society (2004). *SWEBOK. Guide to the Software Engineering Body of Knowledge*. Online at: http://www.swebok.org/.

[60] Iglesias C., Garijo M., Gonzales J., and Velasco J. R. (1998). Analysis and Design of Multi-agent Systems using MAS-CommonKADS. In: *Intelligent Agents IV, Proc. of the Fourth International Workshop on Agent Theories, Architectures, and Languages (ATAL'97)*, Lecture Notes in Artificial Intelligence 1365, Springer-Verlag, pp. 313–326.

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

[62] Jacobson I. (1992). *Object-Oriented Software Engineering*, Addison-Wesley.

[63] Jennings N.R. (2000). On Agent-based Software Engineering. *Artificial Intelligence 117*(2), pp. 277—296.

[64] Juan T., Pearce A., and Sterling L. (2002). ROADMAP: Extending the Gaia Methodology for Complex Open Systems. In: *First International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS 2002)*, ACM Press, pp. 3—10.

[65] Kendall E. A. (2000). Role Modeling for Agent System Analysis, Design, and Implementation. *IEEE Concurrency.* Volume 8 , Issue 2. pp. 34-41.

[66] Knublauch H. (2002). Extreme Programming of Multi-Agent Systems. In: *First International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS 2002)*, ACM Press, pp. 704—711.

[67] Levesque H. J., Reiter R., Lespérance Y., Lin F., and Scherl, R. B. (1997). GOLOG: A Logic Programming Language for Dynamic Domains. *Journal of Logic Programming* 31 (1-3), pp. 59–83.

[68] Lind J. (2001). *Iterative Software Engineering for Multiagent Systems: The MASSIVE Method.* Lecture Notes in Computer Science 1994, Springer-Verlag.

[69] Luck M., and d'Inverno M. (2001). A Conceptual Framework for Agent Definition and Development. *The Computer Journal* 44(1), pp. 1—20.

[70] Luck, M., Ashri, R., D'Inverno, M. (2004). *Agent-Based Software Development*. Artech House Publishers.

[71] McCarthy J., and Hayes P.J. (1969). Some Philosophical Problems from the Standpoint of Artificial Intelligence. In: *Machine Intelligence 4*, Edinburgh University Press, pp. 463—502.

[72] McIlraith S., Son T. C., and Zeng, H. (2001). Semantic Web Services. *IEEE Intelligent Systems* 16(2), pp. 46-53.

[73] Miller J., and Mukerji, J. (eds) (2003). *MDA Guide Version 1.0.1,* omg/2003-06-01.

[74] Molesini, A., Omicini, A., Ricci, A., and Detti, E. (2005). Zooming Multi-Agent Systems. In: *6th International Workshop Agent-Oriented Software Engineering (AOSE 2005),* pp. 193-204.

[75] Newell A. (1982) The Knowledge Level, *Artificial Intelligence*, 18, pp. 87–127.

[76] Odell J. (2002) Objects and Agents Compared. *Journal of Object Technology* 1(1), pp. 41—53.

[77] OMG (2000). *Agent Technology – Green paper*, Agent Platform Special Interest Group, OMG Document agent/00-09-01, version 1.0, 1 September 2000, http://www.objs.com/agent/index.html.

[78] OMG (2000). *Mobile Agent Facility, version 1.0.* OMG Document - formal/00-01-02, online at http://www.omg.org/cgi-bin/doc?formal/2000-01-02.

[79] Omicini A. (2001). SODA: Societies and Infrastructures in the Analysis and Design of Agent-Based Systems. In: Agent-Oriented Software Engineering: First International Workshop, AOSE 2000. Lecture Notes in Computer Science 1957, Springer-Verlag, pp. 185—193.

[80] Omicini A., Papadopoulos, G. A. (2001). Why Coordination Models and Languages in AI?. *Applied Artificial Intelligence 15(1)*, pp. 1—10 .

[81] Pavón J., and Gómez-Sanz J. (2003). *Agent-Oriented Software Engineering with INGENIAS.* In: *Multi-Agent Systems and Applications III, 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS'03),* Lecture Notes in Computer Science 2691, Springer Verlag, pp. 394-403.

[82] Pavón J., Gómez-Sanz J. and Fuentes, R. (2005). The INGENIAS Methodology and Tools. In: [55], chapter IX, pp. 236—276.

[83] Pena J., and Corchuelo R. (2005). Towards clarifying the importance of interactions in agent-oriented software engineering. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial* 25 (1), pp. 19-28.

[84] Peña J., Corchuelo R., and Arjona J. L. (2003). A Top Down Approach for MAS Protocol Descriptions. In: *ACM Symposium on Applied Computing SAC'03*, ACM Press, pp. 49-54.

[85] Pressman Roger S. (1982). *Software Engineering: A Practitioner's Approach*, McGraw-Hill Series in Software Engineering and Technology, McGraw-Hill, New York, 6th edition.

[86] Ralyte J., and Rolland C. (2001). An Approach for Method Reengineering. In: *Proc. Conceptual Modeling - ER 2001, 20th International Conference on Conceptual Modeling*, Lecture Notes in Computer Science 2224, pp. 471—484.

[87] Rao A.S., and Georgeff M. P. (1995). BDI Agents: from Theory to Practice. In: *Proc. of the First International Conference on Multi-Agent Systems (ICMAS'95)*, The MIT Press, pp. 312-319.

[88] Ricci A., and Omicini A. (2003). Supporting Coordination in Open Computational Systems with TuCSoN. In: *12th IEEE International Workshops on Enabling Technologies (WETICE 2003), Infrastructure for Collaborative Enterprises.* IEEE Computer Society, pp. 365—370.

[89] Ricci A., Omicini A. and Denti E. (2003). Activity Theory as a Framework for MAS Coordination. *Engineering Societies in the Agents World III, 3rd international Workshop (ESAW'02),* Lecture Notes in Computer Science 2577, Springer-Verlag, pp. 96—210 .

[90] Rumbaugh J., Jacobson I., and Booch, G. (1999). *The Unified Modeling Language Reference Manual*. Addison Wesley. Reading, MA.

[91] Russell S., and Norvig P. (1995). *Artificial Intelligence; A Modern Approach*. Englewood Cliffs, NJ: Prentice Hall.

[92] Schreiber A., Wielinga J., Akkermans J., and de Velde W. V. (1994). *CommonKADS: A Comprehensive Methodology for KBS Development*. Technical report, Univ. of Amsterdam, Netherlands Energy Research Foundation ECN and Free Univ. of Brussels.

[93] Schillo, M., and Fischer, K. Holonic Multiagent Systems. *Zeitschrift für Künstliche Intelligenz*, no. 3 (in printing).

[94] Serrano J. M., and Ossowski S., (2004). On the Impact of Agent Communication Languages on the Implementation of Agent Systems. In: *Cooperative Information Agents VIII, 8th International Workshop, CIA 2004,* Lecture Notes in Computer Science 3191, Springer-Verlag, pp. 92—106.

[95] Sierra C., Rodríguez-Aguilar J.A., Noriega P., Esteva M., and Arcos J.L. (2004). Engineering Multi-agent Systems as Electronic Institutions. *Upgrade, The European Journal for the Informatics Professional, V(4)*, pp. 33—39.

[96] Singh M. (1997). Formal Methods in DAI: Logic Based Representation and Reasoning. In: *Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence*, pp. 331–376.

[97] Sommerville I. (2004). *Software Engineering 7th edition.* Addison Wesley.

[98] Spivey J. (1992). *The Z Notation: A Reference Manual*. Prentice Hall, Hemel Hempstead, 2nd edition.

[99] Sturm, A., Dori, D., and Shehory, O. (2003). Single-Model Method for Specifying Multi-Agent Systems. In: *Proceedings of the Second International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2003)*, ACM Press, pp. 121—128.

[100] Sturm A., and Shehory O. (2004) A Framework for Evaluating Agent-Oriented Methodologies. In: *Agent-Oriented Information Systems, 5th Int. Bi-Conference Workshop, AOIS 2003*. Lecture Notes in Computer Science 3030, Springer-Verlag, pp. 94—109.

[101] Tavares da Silva J.L., and Demazeau Y. (2002). Vowels Co-ordination Model. In: *First International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS 2002)*, ACM Press, pp. 1129—1136.

[102] Tolvanen, J.-P., and Lyytinen, K. (1993) Flexible Method Adaptation in CASE - the Metamodeling Approach. *Scandinavian Journal of Information Systems*, Vol. 5. IRIS Association. pp. 51-77.

[103] van Lamsweerde A. (2001). Goal-Oriented Requirements Engineering: A Guided Tour. In: *Proceedings of the 5th IEEE International Symposium on Requirements Engineering (RE 2001)*, IEEE Computer Society, pp. 249.

[104] Wagner G. (2003). The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior, *Information Systems* 28 (5), pp. 475—504.

[105] Winograd T., and Flores C.F. (1986). *Understanding Computers and Cognition: A New Foundation for Design*. Norwood, NJ: Ablex.

[106] Wooldridge M., and Ciancarini P. (2001). Agent-Oriented Software Engineering: The State of the Art. In: *Agent-Oriented Software Engineering: First International Workshop, AOSE 2000*. Lecture Notes in Computer Science 1957, Springer-Verlag, pp. 1—28.

[107] Wooldridge, M. (2000). *Reasoning about Agents*. The MIT Press, Cambridge, MA.

[108] Wooldridge M., Fisher M., Huget M.-P., and Parsons S. (2002). Model Checking Multi--agent Systems with MABLE. In: *First International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS 2002)*, ACM Press, pp 952—959.

[109] Wooldridge M., Jennings N. R., and Kinny, D. (2000). The Gaia Methodology for Agent-Oriented Analysis and Design. *Journal of Autonomous Agents and Multi-Agent Systems* 3(3), pp. 285-312.

[110] Yu E. (1997). Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In: *Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering (RE'97)*, pp. 226—235.

[111] Zambonelli F., Jennings N., and Wooldridge M. (2003). Developing Multiagent Systems: the Gaia Methodology. *ACM Transactions on Software Engineering and Methodology* 12(3), pp. 417-470.

[112] Zambonelli F. and Jennings N. R., Omicini A. and Wooldridge M. (2001). Agent-Oriented Software Engineering for Internet Applications. In: *Coordination of Internet Agents: Models, Technologies, and Applications*, Springer Verlag, pp. 326-346.

Last access date for web links reported in the paper: 30-08-2005