The Knowledge Engineering Review, Vol. 30:4, 375–393. © Cambridge University Press, 2015
doi:10.1017/S0269888915000053

Understanding Agent-Oriented Software Engineering methodologies

JORGE J. GÓMEZ-SANZ and RUBÉN FUENTES-FERNÁNDEZ

GRASIA Research Group, Universidad Complutense de Madrid, Avda. Complutense, 28040 Madrid, Spain;
e-mail: jjgomez,ruben@fdi.ucm.es

Abstract

For many years, the progress in agent-oriented development has focused on tools and methods for particular development phases. This has not been enough for the industry to accept agent technology as we expected. Our hypothesis is that the Agent-Oriented Software Engineering (AOSE) community has not recognized the kind of development methods that industry actually demands. We propose to analyze this hypothesis starting with a more precise definition of what an AOSE methodology should be. This definition is the first step for a review of the current progress of an illustrative selection of methodologies, looking for missing elements and future lines of improvement. The result is an account of how well the AOSE community is meeting the software lifecycle needs. It can be advanced that AOSE methodologies are far from providing all the answers industry requires and that effort has grounded mainly in requirements, design, and implementation phases.

1 Introduction

Luck *et al*. (2005) pointed in the last AgentLink roadmap at the lack of mature software development methodologies for agent-based systems as the fundamental obstacle to the take-up of agent technology. They recommended to shift the focus to business issues, looking for quality and convergence with existing and emerging industrial technologies rather than innovation. One may think that Agent-Oriented Software Engineering (AOSE) has improved in these aspects since 2005. To some extent, its methodologies have done it, as it will be shown in this paper, but slowly and without a clear aim. In fact, choosing the right direction to evolve is a challenge in itself.

Some authors advocate by incorporating to AOSE methodologies support for the most recent technologies that are applied in the industry. For instance, Georgeff (2009) proposes actions to integrate AOSE into mainstream (i.e. industrial) Software Engineering (SE). He specifically mentions the integration of *Service-Oriented Architectures* by taking advantage of some intuitive relationships with Multi-Agent Systems (MAS) and AOSE. Georgeff's point, as that of other researchers, is focusing on concepts and methodologies, and not on new specific languages. AOSE concepts and methodologies should drive extensions of current solutions, like the *Service-Oriented Architecture*, rather than intend to replace them. Despite of agreeing with part of this message, we do not think the goal is to take individual technologies and start thinking on agent-oriented extensions. AOSE already tried this, looking for suitable extensions of well-known development methods. The best known case is MAS-CommonKADS, an extension of the successful CommonKADS by Iglesias *et al*. (1997), though more can be found in the same paper. Despite these initial efforts, there is no industrial uptake, yet.

In our opinion, the problem is the AOSE way of developing software. We need to improve it, and this improvement is not achieved by merely adopting one or several paradigms or targeting at concrete technological solutions. Therefore, we bet for a more thorough analysis of AOSE practices, trying to establish a reference model aligned with general SE areas. Such model should be useful to identify weaknesses of current approaches. These weaknesses correspond to SE areas that require more research and that are a concern for the industry. This goal implies a self-criticism for evaluating what kind of methodologies AOSE is building, and what practices are to be continued and what are missing. In order to identify relevant practices in SE, this paper looks at its widely accepted bodies of knowledge, such as the IEEE glossary (IEEE, 1990) and the Software Engineering Book of Knowledge (SWEBOK; Abran *et al.*, 2004).

The analysis starts assuming there is a conceptual gap between AOSE methodologies and industrial SE methodologies. Sturm and Shehory (2003) have raised this issue in the past, though this has not led the AOSE community to self-criticism. As a result, some papers claim to have a methodology if there is a modeling language together with some guidelines. Such cases are closer to a method than to a methodology, a subtle but important difference. Quoting Brinkkemper (1996), in the context of *Method Engineering*, 'the misuse of the term methodology standing for method is a sign of the immaturity of our field, and should consequently be abandoned'. Henderson-Sellers and Giorgini (2005), in the introduction of their edited book, agree with other authors in that a methodology has two components: a first one that describes the process elements of the approach, and a second one focused on the work products and their documentation. They also point out at the common mistake of thinking the second element alone constitutes a methodology. Required elements to build a methodology go beyond that, including, for instance, standards, metrics, and norms. In the same book, Padgham and Winikoff (2005) bring back the issue again, justifying a methodology cannot be only a notation for describing designs or a high-level process. In their case, they choose to include concepts, processes, notations, and guidelines for carrying out the steps of a MAS development.

This mismatch is relevant because if AOSE produces methodologies that do not pay attention to aspects identified as important in SE, industry will certainly tend to consider such methodologies as unrealistic and naïve. So, it becomes utterly important to start from scratch questioning what is an AOSE methodology according to SE shared practices. If a definition existed that could be accepted by software engineers, then compliant methodologies produced by AOSE would have more opportunities to convince the industry, either by replacing, complementing, or extending existing approaches.

The rest of the paper details the points in this introduction and includes the already mentioned study. Section 2 suggests a possible definition for AOSE methodology. Using this definition, we have looked at works claiming to be methodologies and made an assessment of its current state in Section 3. The result can be used as indicator of how close, or far, we are from industry. It can also serve to better communicate to the industry what we are proposing and how it can be used. Continuing with the ideas extracted from the SE review and the analysis, Section 4 discusses some perspectives on the future of AOSE methodologies. This section includes as well further cross-comparison of analyzed methodologies. Section 5 discusses some final conclusions about the findings in the paper.

2 A definition of agent-oriented methodologies from an SE perspective

Talking about the industry implies talking about SE. SE has been around for more than four decades, officially since (Naur and Randell, 1968). It is definitely a field different from Artificial Intelligence, but also from Computer Science (Parnas, 1999). SE aims at the application of engineering to software, that is, 'the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software' (IEEE, 1990). There is a huge collective effort made by several organizations and individuals to compile the Software Engineering Book of Knowledge or SWEBOK (Abran *et al.*, 2004), which summarizes key concepts in SE. This SWEBOK takes into account several standards on SE, from which we highlight the IEEE Glossary for SE (IEEE, 1990). Nevertheless, it does not have a standardized definition of methodology. For this reason, as starting point for our work, Definition 1 adapts the entry of the Oxford dictionary stating that a *methodology* is a system of *methods*:

DEFINITION 1 *An SE methodology is a system of SE methods.*

Since it is a *system of methods*, it should be assumed there are some dependencies and interactions among these methods. Just grouping them does not imply a methodology is being created.

Methods are not defined in IEEE (1990), but referred indirectly as 'the characteristics of the orderly process or procedure used in the engineering of a product or performing a service'. This is in line with the SWEBOK (Abran *et al*., 2004), whose text Definition 2 uses as basis:

DEFINITION 2 *SE methods are those elements imposing structure on the SE activity with the goal of making the activity systematic and ultimately more likely to be successful. Methods usually provide:*

- *a notation and vocabulary*;
- *procedures for performing identifiable tasks*;
- *guidelines for checking both the process and the product.*

Methods vary widely in scope, from a single lifecycle phase to the complete software lifecycle (i.e. since software is conceived to its retirement). They can be: heuristic (structured, data-oriented, or object-oriented), formal (focusing on specification languages, specification refinement, or verification of properties), or prototyping (minding the prototyping style, prototyping target, and the prototyping evaluation techniques).

It is worth noting that there is a distinction between the software development process and the method. The process defines and organizes development activities following some process paradigm, such as Scrum, the Unified Process, or the Waterfall Model. Then, the method provides the assistance for the execution of the activities with notation and vocabulary, procedures, and guidelines, as Definition 2 shows.

The purpose of tools is strongly related with methods, as Definition 3 states:

DEFINITION 3 *Tools are often designed to support particular SE methods, reducing any administrative load associated with applying the method manually. Like SE methods, they are intended to make SE more systematic, and they vary in scope from supporting individual tasks to encompassing the complete lifecycle.*

Definitions 1, 2, and 3 provide the basis for a definition of AOSE methodology:

DEFINITION 4 *An AOSE methodology is a system of AOSE methods.*

The AOSE method is a specialization of the SE method from Definition 2. A mixture of agent- and non-agent-oriented methods can lead to hybrid methodologies. Though interesting, such combinations are out of the scope of this paper:

DEFINITION 5 *An AOSE method is an SE method where the main concept is the Software Agent. In this method:*

- *The notation and vocabulary are based on or reuse concepts from agent research.*
- *There has to be procedures for executing identifiable tasks. These tasks pursue products that realize the agent orientation or parts of it.*
- *There are guidelines for checking the products. These guidelines have also present the agent orientation: as the method concerns agent concepts at least partly, its products also need to do it.*
- *The nature of this method is mainly heuristic, as it relies on features inherent to software agents, for example, adaptiveness or autonomy, to guide the development and structure the system. Nevertheless, such features may also be regarded from a formal perspective, using some mathematical apparatus and specification models to derive products or just to verify properties.*
- *There are support tools that assist in the execution of the different activities addressed by the method. Therefore, if there is some agent bias in the notation and vocabulary, tools must consider it.*

Any method needs a scope which can be the whole software lifecycle (i.e. since software is conceived to its retirement) or a part of it (IEEE, 1990). When it is a part, we define it using a cataloge of the phases likely to be found. These are not activities, but closer to general aspects that an industrial project tends to consider:

DEFINITION 6 *There are some recurrent phases in the lifecycle of software. The way they are addressed, or how they occur, depends on the chosen development process. The phases are: concept* *(viability of the project), requirements (definition of the problem to solve), design (specification of the solution to the problem), implementation (construction of the solution), test (validating the solution), installation and checkout (integrating the product in the production environment), manufacturing (adapting a basic version of the product to the needs of target customers), operation and maintenance (using the product and dealing with new problems), and retirement (support for the removal of the product from production).*

It is worth to conclude reminding that Definition 5, as Definition 2, distinguishes the method and the process. The process supplies the activities, and the method instructs how to perform them, which is the usual way in SE. There has been some tradition in AOSE methodologies to include as well *development activities*. In most cases, such activities are rather simple and would be better regarded as the *identifiable tasks* Definition 2 refers to, that is, concrete instructions to accomplish an activity defined by the development process.

3 Analyzing current methodologies

A combination of Definitions 5 with 6 provides enough elements to analyze the concrete utility of most of the contributions from AOSE in the generic phases of a software lifecycle. As industry applies SE, this exercise gives a hint on the extent to which AOSE methodologies would provide assistance in an hypothetical industrial development.

This analysis has several goals. First, it tries to organize the results of the different approaches by development phases and type of support, in order to facilitate further comparison in future works. Second, it tries to track the main lines of evolution of AOSE methodologies. Here, the analysis considers two groups. On the one hand, we are interested in the evolution of the most long-standing AOSE methodologies, which have been obtained the book (Henderson-Sellers and Giorgini, 2005). This book covers 10 research works that have been continued until the moment of this survey. The foundations of these *established* AOSE methodologies may not be the best ones, though. To include new perspectives on the AOSE area, a selection of additional and more recent methodologies has also been made. Section 3.1 presents the analysis of the first group, that is the *established methodologies*, and Section 3.2 that of the second group, that is, the *new methodologies*.

Several tables summarize the features of these approaches. Their grouping attends only to page layout optimization. Each table has four columns derived from the definitions Section 2 introduced.

The first one contains the name of the methodology and a key reference to it. The second column indicates software lifecycle phases that are addressed by the methodology. A methodology can support several phases. Each phase appears in a different row and it contains the activities or tasks the methodology associates to it. Note that, despite claims of the methodology under study, this analysis considers that a methodology addresses a phase when it includes at least one activity or task for it. In some way, the number of activities may indicate the degree to which the phase is covered. As there is no *analysis phase* in Definition 6, analysis activities/tasks are considered to be either in the requirements or in the design phase. The third column contains the names of the support tools used in each phase. They can be generic tools or specific ones. The fourth column introduces the vocabulary and notation. Tables mainly mention the existence of diagrams drawn following some notation, but formal methods appear as well. Sometimes, there are no diagrams or formal specifications, but just documents. In those cases, the notation is classified as structured language (e.g. tables or forms) or non-structured language (e.g. free text).

For each methodology, the analysis also studies the kind of contributions (i.e. guidelines or tasks) it makes. Because of space limitations, tables do not include explicitly this information, but refer to it indirectly. Tables include the scope of the method, enumerating the *identifiable tasks* Definition 5 mentions. This enumeration implies the consulted sources do explain how these tasks are to be executed and what it is expected from them. If consulted sources do not include either procedures or guidelines related to any identifiable task, the tasks field will be empty. The agent orientation in the products is assumed in every case, so it will not be studied.

The conclusion of this analysis is that the *established methodologies* satisfy better the Definition 4 of an AOSE methodology. There are differences in the scope, being GAIA and MESSAGE the ones assisting the least. PASSI, Tropos, ADELFE, Prometheus, and INGENIAS are coping with most of the software lifecycle, after some years of improvements. The three more active have been Tropos, Prometheus, and INGENIAS, with contributions opening new phases up to 2010. MaSE is being reactivated as O-MaSE with new works. On the other side, the *new methodologies* are better regarded as AOSE methods, because the assistance they provide (in terms of notation, vocabulary, procedures, guidelines, and tools) is sufficient only in concrete phases, mostly design. Current tools for defining vocabularies and notations are more precise and versatile. This may be a reason for the growth of visual modeling languages and the focus on design, where they play a main role. Time will tell if the *new methodologies* evolve from methods to methodologies. We think they will do, at least, in a combination of design and implementation AOSE methods, just like the *established methodologies* started.

Anyway, the analysis shows that AOSE is still far from industrial practices, if we assume these are the ones SE talks about. There is enough experience in requirements and design, but implementation of specifications is not that satisfactory. Sometimes, implementation is addressed with automated model transformations, but then, issues like roundtrip engineering (i.e. moving back changes in the code to the specification) are hardly addressed. Manual coding is not well explained in general or it is straightforward only when particular frameworks are used. Other phases, such as testing, installation/operation, and maintenance, are only recently starting to be addressed. It may not be the only reason, but it is hard to sell a technology whose lifecycle is not completely understood and experienced unless it is the next hype, and agents are no longer.

3.1 Established methodologies

Table 1 corresponds to two of the earliest methodologies to be cited in the literature: GAIA (Zambonelli *et al.*, 2005) and MESSAGE (Garijo *et al.*, 2005). These pioneering works have had a remarkable influence in AOSE.

GAIA has been published as a methodology without support tools. It has the merit of pointing out key concepts that became the base for other works. However, according to Definition 3, this lack of support tools makes GAIA less disciplined than other methodologies. After all, products that are generated without tool assistance are more likely to contain errors. Other works reusing or extending GAIA concepts propose their own tools, such as MASDK (Gorodetsky *et al.*, 2008).

Table 1 Analysis of GAIA and MESSAGE methodologies as introduced in Henderson-Sellers and Giorgini (2005)

Methodology	Scope (phase: tasks)	Tools	Notation/vocabulary
GAIA (Zambonelli *et al.*, 2005)	*Requirements*: analysis		Structured language notation: environmental model, preliminary role model, preliminary interaction model, organizational rules
	Design: architectural design, detailed design		Structured language notation: role model, interaction model, agent model, services model, organizational structure
MESSAGE (Garijo *et al.*, 2005)	*Requirements*: analysis process based on level abstraction	Metaedit + (Kelly *et al.*, 1996)	MESSAGE meta-model
	Design: high level design process, detailed design process, organization-driven detailed design process, agent-platform detailed design process	Metaedit + , UML tools	MESSAGE meta-model, UML diagrams
	Implementation: mappings from detailed design to code (Massonet *et al.*, 2002)	Any IDE	Preliminary implementation

GAIA declares some analysis tasks that could be regarded as part of the requirements phase (because they deal with the identification of non-functional requirements) or the design phase (because they identify elements that are part of the solution). In both cases, there are brief guidelines to check the specifications and procedures for creating them in these two phases. Therefore, GAIA can be regarded as a combination of requirements and design AOSE methods. GAIA could be an AOSE methodology according to Definition 4, but it is the only one that does not regard implementation phases, which is infrequent.

MESSAGE (Garijo *et al.*, 2005) is a methodology with a wider scope than GAIA. It was a pioneer work in providing a notation and vocabulary using meta-models in MAS, having only AALAADIN (Ferber & Gutknecht, 1998) as precedent. A meta-model serves usually describes the abstract syntax of a modeling language, that is, its key elements and their relationships. The notation corresponds to the way concepts of the meta-model are drawn (i.e. concrete syntax) and it is frequently described apart. MESSAGE tools are based on MetaEdit+ (Kelly *et al.*, 1996). This tool is a generic meta-editor that takes as input the meta-model of a visual modeling language and produces as output editors implementing them. The value of the MESSAGE meta-model for the developer is twofold. It provides a precise vocabulary for specifying the system and facilitates the task of declaring how to implement it. MESSAGE also provides guidelines and procedures to create the diagrams it identifies, to derive a low-level design, and to obtain implementations. MESSAGE had several proof-of-concept implementations, but only one was publicly released (Massonet *et al.*, 2002). This implementation suggests how meta-model elements could be mapped into implementation constructs. MESSAGE is a good example of the benefits of having tool support, for the developer and for the sake of building new knowledge. Using again Definitions 4 and 5, it can be regarded as an AOSE methodology. There is a system of methods as its requirements, design, and implementation AOSE methods have mutual dependencies.

Table 2 introduces a second group of *established methodologies*: PASSI (Cossentino, 2005), Tropos (Giorgini *et al.*, 2005a), and Prometheus (Padgham & Winikoff, 2005). They have a broader scope than GAIA and MESSAGE, since they have been continuously improved along the years. Also, each considered phase has at least a support tool.

Tropos is the one that has been around for a longer time. This is one of the reasons for it having the largest number of tools and related published works. Regarding notation and vocabulary, it is based on *i** (Yu, 1997), so it has fewer and simpler, yet expressive, diagrams compared with other methodologies. It has succeeded to become a reference in SE, particularly in requirements modeling. Its support tool, TAOM4E (Morandini *et al.*, 2008), is evolving toward a meta-model-driven infrastructure, similar to that MESSAGE used. TAOM4E is based on Eclipse. Coding in Tropos is a more complex task than in the other methodologies of this table, that is, Prometheus and PASSI. In Tropos, the transition from specification to code is completely manual, while Prometheus and PASSI provide some aid for it. Testing is being developed in a separated research line, though connected and validated over Tropos. It is the Goal Analysis Testing method, supported by the eCAT tool (Nguyen *et al.*, 2010). Tropos has enough guidelines and procedures for each one of the identified tasks. In fact, each contribution is supported by individual papers and many case studies. Definition 4 is straightforward to apply in this case, since it is truly a system of AOSE methods what Tropos proposes.

PASSI made an important effort to integrate mainstream SE practices in AOSE. It uses a generic Unified Modeling Language (UML) tool instantiated with a profile for defining its concepts as stereotyped classes. The reuse of a UML tool allows working at the class level and performing mappings easily with concepts close to implementation. PASSI concerned specially code reuse, an activity initially associated to the implementation phase but that could be also to the maintenance one. The patterns identified and automatically coded with its Agent Factory tool may serve both for implementation, re-engineering, or reverse engineering of the system. PASSI has also been associated to the installation and checkout phase, because it deals with the way the system has to be deployed. PASSI does not mention this explicitly, but we assume that if a coding phase has been performed and a deployment defined, there is a product that can be installed and developers must ensure that such installation is possible. Testing is another concern in PASSI, though little details are given. The work (Caire *et al.*, 2004) introduces a testing framework with the vocabulary to be used when defining tests for PASSI. Like Tropos, PASSI identifies relevant tasks for each phase, though the guidelines and procedures are to be found across several papers. Unlike Tropos,

Table 2 Analysis of PASSI, Tropos, and Prometheus as introduced in Henderson-Sellers and Giorgini (2005)

Methodology	Scope (phase: tasks)	Tools	Notation/vocabulary
PASSI (Cossentino, 2005)	*Requirements*: domain req. description, agent identification, role identification, task specification	PTK (Cossentino *et al*., 2003)	UML notation with concrete stereotypes: System Requirements Model
	Design: ontology description, role description, protocol description, Agent structure definition, Agent behavior description	PTK	UML notation with concrete stereotypes: agent society model, agent implementation model
	Implementation: code reuse, code production	PTK, AgentFactory (Sabatucci *et al*., 2006), JADE	Code model
	Test: agent test		Testing framework instantiation (Caire *et al*., 2004)
	Installation and checkout phase: deployment configuration	PTK	UML notation with concrete stereotypes: deployment model
Tropos (Giorgini *et al*., 2005a)	*Requirements*: early requirements, late requirements	TAOM4E (Morandini *et al*., 2008), T-Tool (Fuxman *et al*., 2001) + GR-Tool (Giorgini *et al*., 2005b)	*i**-based notation: actor diagram, rationale diagram/formal Tropos
	Design: architectural design, detailed design	TAOM4E, UML compliant tools	*i**-based notation: actor diagram, rationale diagram UML notation: sequence diagrams, class diagrams, state chart diagrams
	Implementation: guidelines for mapping concepts to agent platforms	TAOM4E, JADE/Jack	
	Test: goal analysis testing (Nguyen *et al*., 2010)	eCAT	
Prometheus (Padgham & Winikoff, 2005)	*Concept*: specification phase	PDT (Padgham *et al*., 2008) + (Sun *et al*., 2010)	Structured text: use case scenarios
	Requirements: specification phase	PDT	Custom notation. Goal diagram, text descriptors for functions, use case scenarios
	Design: architectural design, detailed design	PDT	Custom notation: data coupling diagram, agent acquaintance diagram, AUML diagrams, system overview diagram, agent overview diagram
	Implementation: guidelines (Padgham & Winikoff, 2004), automatic code generation for Jack	Custom PDT to Jack plugin, Jack development Environment, Jadex	
	Test: automatic generation of tests (Zhang *et al*., 2011)	PDT	Standard test description

PASSI has not made an effort to maintain a central site grouping together its different AOSE methods. Anyway, we think it satisfies as well Definition 4.

Finally, Prometheus is a well-known work, with an adequate coverage of basic software lifecycle phases. Its main support tool is the Prometheus Design Tool (PDT; Padgham *et al.*, 2008). It has conversion tools that can export specifications made with PDT into Jack Agent Platform projects. Nevertheless, the reverse is not possible automatically, and this makes harder any potential maintenance work. Its notation borrows concepts from UML and Agent UML. Like Tropos, it has migrated its support tools to Eclipse, again following the meta-model approach like MESSAGE. The contribution of Prometheus to the concept phase is rarely found in an agent-oriented methodology. PDT uses text to describe the scenarios, something that could be categorized as the initial steps to determine if the system is viable. Nevertheless, viability studies are not explicitly made in Prometheus. There are also procedures and guidelines gathered in a book (Padgham & Winikoff, 2004). The result is a very coherent AOSE methodology, well documented and with an adequate coverage of most of the software lifecycle.

The last group of *established methodologies* appears in Table 3. It presents INGENIAS (Pavón *et al.*, 2005), ADELFE (Bernon *et al.*, 2005), and MaSE (DeLoach & Kumar, 2005).

INGENIAS decouples processes and methods. The development process is the Rational Unified Process (RUP). This facilitates the integration of the INGENIAS methods, since it is sticking to standard activities in a previously existing development process, unlike previous works. INGENIAS coverage of the concept phase is linked to the inception phase of the RUP, though it does not suggest any specific method here. The other phases have a more explicit support in INGENIAS, specially the requirements one with the work done in UML for Activity Theory (Fuentes *et al.*, 2006). The installation and checkout phase is considered included in INGENIAS for similar reasons to those for PASSI: both intend to define the way the system is deployed. The maintenance phase is better supported, because there is experience in keeping a development for a long time and practices to make this possible. Concretely, it includes roundtrip engineering techniques to go from the specification to code and back [?]. This allows modifying a system while keeping coherence between the specification and code. Also, it permits to modernize an existing INGENIAS development by using new code generation facilities in old specifications. As MESSAGE, INGENIAS bases its vocabulary and notation on a meta-model (Gómez-Sanz & Pavón, 2002). Regarding tools, INGENIAS does not use Eclipse, unlike other works. It has its own meta-model facilities developed from scratch, the INGENME tool (`http://ingenme.sf.net`). Documentation of INGENIAS is less extensive than that of Tropos or Prometheus, at least in English. There is technical reference material instructing in the kind of intermediate products to be created toward a running MAS. There are additional guidelines and procedures for concrete RUP activities. As a conclusion, it satisfies Definition 4 to be regarded as an AOSE methodology.

ADELFE is another methodology that integrates into the RUP, like INGENIAS. It uses a commercial customized UML tool called Open Tool (Bernon *et al.*, 2005). Open Tool has scripts for translating interactions into simulations of the communication among agents. Because it serves to perform a validation of the specification but not the product itself, it has been regarded as part of the design phase. ADELFE has incorporated Adaptive MAS Modeling Language recently (Rougemaille *et al.*, 2008). This allows defining a MAS more precisely than with Open Tool, and producing some preliminary implementation with the aid of the MAY framework. ADELFE also provides an inexpensive approach for determining the adequacy of using MAS for solving a problem. This has been categorized within the concept phase. It also invests a great deal of effort in identifying meaningful tasks that fit within the RUP process. It provides sufficient guidelines and procedures to perform them. Besides, these are included in a single web site (`http://www.irit.fr/ADELFE/`). Like previous cases, it satisfies Definition 4 enough to be regarded as an AOSE methodology.

MaSE is another classic methodology with a sophisticated support tool, namely agentTool (DeLoach & Wood, 2000). This tool has facilities to perform formal verification on communications using the already existing SPIN model checker. The agentTool automatically translates a MaSE specification to the PROMELA language of SPIN, and launches the necessary verification. As in the case of ADELFE, this has been regarded as part of the design phase, since the verification is applied to the specification, not to the actual system. MaSE uses a notation strongly based on UML, though agentTool is not itself an UML tool, unlike those of PASSI and ADELFE. As early versions of PDT and TAOM4E, agentTool embodies the

Table 3 Analysis of INGENIAS, ADELFE, and MaSE as introduced in Henderson-Sellers and Giorgini (2005)

Methodology	Scope (phase: tasks)	Tools	Notation/vocabulary
INGENIAS (Pavón *et al.*, 2005)	Concept: inception phase with basic information gathering	UML tools, IDK, IAF	UML diagrams, INGENIAS meta-model
	Requirements: analysis activities in inception, elaboration and construction phases, requirements elicitation with UML-AT (Fuentes *et al.*, 2006)	IDK (Gómez-Sanz *et al.*, 2008)	INGENIAS meta-model, UML-AT diagrams
	Design: design activities in inception, elaboration and construction phases	IDK	INGENIAS meta-model
	Implementation: automatic code generation and manual coding (Gómez-Sanz & Pavón, 2005; Gómez-Sanz, 2007; García-Magariño *et al.*, 2008)	IDK, IAF	Code templates and specification traversal techniques (Gómez-Sanz, 2007)
	Test: debugging/testing guidelines (Gómez-Sanz *et al.*, 2009)	IDK, IAF	Meta-model extensions for testing and simulation (Gómez-Sanz *et al.*, 2010)
	Installation and checkout phase: deployment definition (Gómez-Sanz *et al.*, 2009, 2010) development vs. production versions of constructed MAS (Gómez-Sanz, 2007)	IDK, IAF	Instructions for launching applications in production mode (Gómez-Sanz, 2007)
	Maintenance: roundtrip engineering practices (Pavón *et al.*, 2006)	IDK, IAF	INGENIAS meta-model
ADELFE (Bernon *et al.*, 2005)	*Concept*: verify the AMAS adequacy	AMAS adequacy tool	
	Requirements: define user requirements + validate user requirements, define consensual requirements, establish keywords set, extract limits and constraints, characterize environment, determine use cases, elaborate UI prototypes, validate UI prototypes	OpenTool (Bernon *et al.*, 2005), interactive tool (Bernon *et al.*, 2005)	UML notation and free text documents. Preliminary requirements, final requirements
	Design: analyze the domain, identify agents, study interactions between entities + study detailed architecture and multi-agent model, study interaction languages, design agents, complete design diagrams, protocol simulation	Open Tool, scripts included in Open Tool, AMAS ML (Rougemaille *et al.*, 2008)	UML notation with concrete stereotypes, MAS description with AMAS ML
	Implementation: fast prototyping	MAY (Rougemaille *et al.*, 2008)	Agent code
MaSE (DeLoach & Kumar, 2005)	*Requirements*: analysis phase activities	agentTool (DeLoach & Wood, 2000)	Custom notation combined with UML notation: goal hierarchy, use cases, sequence diagrams, concurrent tasks, role model
	Design: design phase activities, formal verification of protocols	agentTool, SPIN	Custom notation combined with UML notation: agent class diagrams, conversation diagrams, agent architecture diagrams, deployment diagrams/SPIN's PROMELA
	Implementation: code generation	agentTool, agentTool III (García-Ojeda *et al.*, 2009)	

notation of the methodology, though it has been recently migrated to an Eclipse-based version. This new version is called agentTool III (García-Ojeda *et al.*, 2009). It offers better support for the implementation phase than its predecessor. It has automated code generation from the specification, though not yet at the level required to facilitate maintenance. MaSE also regards deployment diagrams, but, in this case, it does not score in the installation and checkout phase. The reason is that, despite talking about deployment, there is not yet any realization of the deployment diagram in software, unlike in the other cases, where implementation activities are more mature. Documentation on procedures and guidelines was included in a site for the initial versions of MaSE (`http://macr.cis.ksu.edu/mase`) and is also for the new version O-MaSE (`http://macr.cis.ksu.edu/omase`). Like Prometheus, Tropos, and PASSI, MaSE proposes its own development process intertwined with the method. Anyway, we think it sufficiently satisfies Definition 4 to be regarded as an AOSE methodology.

3.2 New methodologies

The book (Henderson-Sellers & Giorgini, 2005) includes a convincing selection of methodologies existing up to 2005. To account for new works after this year, the AOSE workshop proceedings have been inspected. From the different published papers, several have been chosen that meet the three following conditions: to claim a new methodology is presented; to have continued the work in other papers; and being supported by tools. The selected works are ASEME (Spanoudakis & Moraitis, 2011), OperA (van Putten *et al.*, 2009), GorMAS (Argente *et al.*, 2008), Dsml4mas (Hahn & Fischer, 2008), and ForMAAD (Graja *et al.*, 2010). Table 4 shows the results of their analysis. Their scope is more reduced than that of previous works reviewed in Section 3.1. Besides, guidelines or concrete tasks are missing in some cases.

ASEME (Spanoudakis & Moraitis, 2011) stands for *Agent Systems Engineering Methodology*. It uses model-driven techniques to create MAS. The notation and vocabulary are specified using meta-models with the Eclipse Modeling Framework (EMF). The resulting modeling language is called the Agent MOdeling LAnguage (AMOLA; Spanoudakis & Moraitis, 2008). AMOLA is made in fact of several meta-models, each one devoted to a specific development stage. The scope of the methodology covers the requirements, design, and implementation phases. Transitions from one to the other are made through transformations written in the ATLAS Transformation Language (ATL). From design to code, it uses IAC2JADE (Spanoudakis & Moraitis, 2010), which transforms specifications into message sending/receiving behaviors. Guidelines are missing since authors rely just on automated model transformations. The same happens with the tasks to be performed in each phase of the development. As a collection of AOSE methods, ASEME does not include some critical elements pointed out by Definition 5.

GorMAS (Argente *et al.*, 2008) covers the requirements and design phases. It defines the activities within each phase using the Software and System Process Engineering Meta-model (Object Management Group, 2008) and describing the result of the activities (Argente *et al.*, 2011b). The different models involved are represented using the GorMAS meta-models (Argente *et al.*, 2008). EMFGormas (García *et al.*, 2010) is the representation of the GorMAS meta-models using EMF and the Graphical Modeling Framework of Eclipse. There exist descriptions of individual diagrams, but guidelines and procedures are missing. Also, like GAIA, it is limited to design and requirements phases, without concerns regarding implementation. Unlike in GAIA, there are some initial support tools, what ensures a greater discipline in the construction of the methods. As a consequence, GORMAS still requires more work to be regarded as an AOSE methodology or some AOSE methods.

OperA is 'a methodology developed to represent and analyze organizational systems' (van Putten *et al.*, 2009). Operetta (Aldewereld & Dignum, 2010) is the graphical environment built in Eclipse. OperA has no specific process, but some recommendations are presented as *design guidelines* (Aldewereld & Dignum, 2010). Despite these guidelines, it is not evident from the published papers, which phases are addressed. There are three models to take into account: Organizational Model, Social Model, and Interaction Model. The Organizational Model is the result of the analysis and provides the organization design that satisfies the requirements. The Social Model defines the constraints under which the organization operates, and could be obtained as well during a requirements phase. The Interaction Model contains the agreements among role enacting agents. Taking into account the level of detail supplied by these models,

Table 4 New methodologies proposed in the AOSE workshop since 2005 edition

Methodology	Scope (phase: tasks)	Tools	Notation/vocabulary
ASEME (Spanoudakis & Moraitis, 2011)	*Requirements*: no guidelines for producing the models. Driven by a waterfall of transformations	UML tools + ATL	AMOLA (Spanoudakis & Moraitis, 2008) using UML notation: use case diagrams, Agent Interaction Protocols, Roles model
	Design: no guidelines for producing the models. Driven by a waterfall of transformations	UML tools + ATL	AMOLA (Spanoudakis & Moraitis, 2008) using state charts: inter-agent control model, intra-agent control model
	Implementation: no guidelines for producing the models. Driven by a waterfall of transformations	IAC2JADE (Spanoudakis & Moraitis, 2010) + ATL	Code for Jade
GORMAS (Argente *et al.*, 2008)	*Requirements*: mission analysis, service analysis	EMFGormas (García *et al.*, 2010)	Meta-model based: organization model and activity model
	Design: organizational design, organization dynamics design	EMFGormas	Meta-model based: organization model, activity model, normative model, interaction model, agent model
OperA (van Putten *et al.*, 2009)	*Requirements*: identify requirements, identify stakeholders, set social norms, redefine behavior, create interaction scripts	OperettA (Aldewereld & Dignum, 2010)	Meta-model based: social diagram editor, Interaction diagram editor, ontology manager, partial state description editor
	Implementation: applying model transformations	OperettA, AgentScape	AgentScape code
DSML4MAS (Hahn & Fischer, 2008)	*Design*: no declared tasks	DDE (Warwas & Hahn, 2009)	DSML4MAS (Hahn & Fischer, 2008) using custom notation: Multi-agent system aspect, agent aspect, organization aspect, interaction aspect, behavior aspect, environment aspect
	Implementation: automatic code generation (Hahn *et al.*, 2009)	DDE, Jack, Jade	Code for Jade or Jack
ForMAAD (Graja *et al.*, 2010)	Requirements: no declared tasks	StarUML	UML notation: requirements specification model
	Design: cooperation strategy definition, organization structure definition, collective behavior definition, individual behavior definition, translation instructions to Z notation	StarUML + XSLT transformations + Z/EVES	AML language using UML notation: role identification diagram, organization structure diagram, precedence order graph, activity diagram, class diagram

they all could be produced during a requirements phase, since the information concerns what the system requires and what actors are needed, but not how the system is going to do it. Nevertheless, authors mention the use of transformations to map the information contained within these models into agent frameworks, concretely AgentScape, but details are not given. In the evaluation of OperA, we focus on the requirements phase OperA seems to cover best, including guidelines and procedures. As a result, we find OperA an AOSE requirements method rather than a methodology.

Dsml4mas (Hahn & Fischer, 2008) is a generic modeling language that declares to be platform independent. It is presented in (Hahn *et al.*, 2009) as 'a methodology-like approach, where a model transformation takes a protocol description and generates a corresponding behavior description'. The Dsml4mas Development Environment (Warwas & Hahn, 2009) is an Eclipse-based editor that implements Dsml4mas using the EMF. As such, it uses the ATL to produce another model following a Jack or Jade platform specific model. These models can be converted into actual code using MOFScript, another EMF-specific tool. Semantics of the Dsml4mas are introduced using Object-Z in Hahn & Fischer (2008). Regarding the considered works, Dsml4mas is not proposing exactly either a method or a methodology, because the procedures and guidelines to build the software system have not been defined yet. As GorMAS, it seems to be building a notation and vocabulary. Unlike GorMAS, it seems more advanced because it is studying how to produce code from the specification.

ForMAAD (Graja *et al.*, 2010) is a reformulation of an older work (Kacem *et al.*, 2007) to use a model-driven approach. Original ForMAAD focused on how the Z notation could be used to specify MAS. The model-driven version instead uses the Agent Modeling Language (AML; Cervenka *et al.*, 2006a, 2006b). AML models are built using StarUML, a generic UML tool, and transformed into Temporal-Z (Regayeg *et al.*, 2004) using eXtensible Stylesheet Language Transformations. The Temporal-Z result is then imported by the Z/EVES tool, a generic Z theorem prover, to prove the necessary theorems. The use of ForMAAD requires first a specification phase, which gathers the requirements specification model. Then, there is a design phase that follows several refinement steps. Using the available information, requirements phase is not correctly supported. There may be notation and vocabulary, and even tool support, but there is no guidance for the developer either to create the requirements model or to check it. Design phase is handled better, with instructions for obtaining the different diagrams and for evaluating the resulting models. Though the paper refers implicitly to implementation, this is actually a transformation of the models proposed by the authors to the Z notation, which we still regard as being a specification, not a coding product. The consequence is that ForMAAD is mainly an AOSE design method.

4 Perspectives for agent-oriented methodologies

The analysis made, though it does not address every published work, allows to confirm a tendency toward the application of model-driven development solutions and an increasing relevance of support tools. This progress will be more thoroughly studied in the following two sections, attending to the software lifecycle phases and the notation/vocabulary. Section 3 considers potential future areas to regard by existing or future methodologies.

4.1 Software lifecycle phases

New works (see Section 3.2) tend to focus on the requirement and design phases, specially the later. The implementation phase is not always present but, when it is regarded, it involves automatic code generation in every case. On the other hand, most of the established methodologies (i.e. PASSI, Tropos, ADELFE, Prometheus, INGENIAS, and MaSE, reviewed in Section 3.1) are aiming to cover most of the software lifecycle. In our opinion, the incorporation of the implementation phase has been critical for this growth, as it permits to progress toward the testing and installation-checkout phases. Though their coverage of the software lifecycle is not complete, we think they deserve more being called methodologies than the new works. As in the case of GAIA and MESSAGE, most of these new works may be more conveniently introduced as AOSE methods. This is the case of GorMAS (focusing on the requirements and design phases), OperA (mainly requirements with some implementation concern), Dsml4MAS (mainly design and implementation, but very little guidance is given), and ForMAAD (focusing on the requirements and design phases). From the selected works, only ASEME seems to be following the evolution path of established methodologies, though it is still work in progress.

Whether the community should aim for *phase-specific methods* or *whole lifecycle methodologies* is a difficult question. In any case, both methods and methodologies should be clearly distinguished. Also, the AOSE community should be capable of integrating its results with industry practices no matter the development phase. This can be done with isolated methods and methodologies. It is the researcher who decides to provide contributions as *AOSE methods*, and then to put them all together into a methodology or not. Others will decide if the resulting methodology is of interest or if it is better to reuse just some phase-specific methods. In our opinion, a methodology covering the whole lifecycle is more convincing, since it has a more coherent view across methods dealing with concrete phases. After all, a methodology is a system of methods (Definition 1).

A positive aspect is the important increase in the use of SE process specification methods to describe the processes associated with methodologies, as in ASEME or GorMAS. This may be the result of an increasing promotion of *Method Engineering* practices in the AOSE community together with the availability of open source tools, like the Eclipse Process Framework Composer for processes definition. Traditionally, development processes have been rarely acknowledged in the AOSE community with some exceptions (Seidita *et al*., 2009). There are methodologies that propose new *software development processes* (e.g. PASSI, Tropos, or Prometheus) while others stick to an existing *software development process paradigm* (e.g. INGENIAS or ADELFE). Understanding the relevance of processes is something the IEEE FIPA Design Process Documentation and Fragmentation Working Group is contributing enormously. Interested readers can find complete specifications of processes for PASSI, GORMAS, INGENIAS, and others at its website (`http://www.fipa.org/subgroups/DPDF-WG.html`). For more information about the role of processes in AOSE, it is recommended to read the survey from Cossentino *et al.* (2011).

A major challenge in the future of AOSE is the problem size. To be accepted, agent technology and AOSE methodologies should prove they can support regular size developments. The adequacy of a development process and methodology is only evident when a sufficiently large problem is considered. So that readers understand, it is not rare to find industrial software projects ranging from tens to hundreds of people working full time. Our AOSE case studies, in most cases, take few people during some months. This is how far we are. Considering real size developments will surely imply addressing seriously some not-so-popular phases, like operation-maintenance or installation-checkout.

4.2 Notation and vocabulary

Important progress has been made in what refers to how AOSE methodologies approach the issue of vocabulary and notation. AOSE has payed attention to formal notations, as shown in El Fallah-Seghrouchni *et al.* (2011), though the focus now seems to be on semi-formal ones. Semi-formal approaches can appear as visual modeling languages specified through meta-models.

The combination of meta-models plus meta-modeling tools is being followed by most of methodologies now. Argente *et al.* (2011a) and this paper itself prove this major trend. Almost all recent works use Eclipse to build visual editors using meta-models as input. Established methodologies are migrating toward this platform too, with TAO4ME (Bertolini *et al*., 2006) for Tropos, agentTool III (Sierra *et al*., 2009) in the case of O-MaSE, and PDT (Sun *et al*., 2010) for Prometheus. INGENIAS has used this meta-modeling approach since its beginning (Pavón & Gómez-Sanz, 2003), publishing its meta-model as an XML file in its distribution site (`http://ingenias.svn.sourceforge.net/viewvc/ingenias/trunk/metamodel/`). As a contribution to the community, the INGENIAS team recently released to the public the meta-editor framework INGENME, that was internally used to maintain INGENIAS. INGENME (available at `http://ingenme.sf.net`) provides similar functionality to Eclipse in terms of visual editors generation.

Advances in the construction of modeling languages have been encompassed with advances in techniques for implementing specifications. A review of them can be found in Nunes *et al.* (2011). The review shows an outstanding growth of solutions based on automatic translation of specifications into code. Most of them reuse facilities from the Eclipse platform, like the already mentioned TAO4ME, agentTool III or PDT. This approach is the dominant now, with some exceptions where implementation has to be done manually.

There are precedents in this kind of manual transition from specification to code. It can be done in a disciplined way, as suggested by Kendall (1999), where developers have the assistance of guidelines like design patterns.

The future of notations and vocabularies for methodologies seems to be associated to meta-models and transformations. Due to the few available alternatives, Eclipse will probably keep on being the major platform for supporting this kind of development. The good news is that Eclipse will release researchers from most of the maintenance of visual editors. This will allow focusing more on the evolution of modeling languages and reducing the costs of experimenting with visual modeling languages.

4.3 Making methodologies better

Despite the improvements, none of the existing methodologies can be considered complete, mature, or highly effective. They need to gain more experience in real developments. This will surely make them evolve further, probably toward one or several of these goals: addressing new development activities that increment the scope of the methodology; improving the tool support; or increasing the number of developments that endorse the possibilities of each methodology.

A successful methodology increases the chances of building successfully a system. Consequently, a methodology needs to be applied and assessed. A way to rate the capability for success is the Capability and Maturity Model (Paulk *et al.*, 1993) and its evolution, the Capability Maturity Model Integration (CMMI Product Team, 2002). These models identify a number of key areas that need to be studied to assess the capability of organizations that develop software. Some of these areas are the technical skills of developers, the degree of adoption of software processes, the existence of evaluation procedures to assess the work done, or the training of personnel within the company. Many of these areas point put to elements that methodologies should consider in order to be regarded as enough mature for industrial development. Our methodologies are not prepared for such evaluation, yet. Only a minimal part of these elements are covered actually, and we are not sure how well they have been.

Tool support will be improved in all the phases. A basic improvement will include being capable to enrich a methodology using past experiences. Notations like UML have changed over the years because the application experience of UML told there were things that could be made better. Similarly, our methodologies should be able to improve. This requires having customizable tools, capable of adapting to new processes or notations. Thanks to the adoption of meta-modeling techniques by methodologies, this kind of tools are available. Changes in the meta-models describing the modeling language or the process will be automatically processed to produce a new tool set adapted to the new meta-models.

Finally, there will be a higher demand of evidences proving a methodology works. These evidences will have to be complete developments endorsing the capability of the methodology. There has been intents to develop the same system with Prometheus, O-MaSE, Tropos, and the MAS Unified Process (Luck & Padgham, 2008). This can be useful to show the pros and cons of each approach and demonstrate their benefits. Also, some methodologies maintain a list of developments that can be used for testing the methodology and learn what it can do. Tropos keeps a list of papers introducing empirical application of the methodology (see `http://www.troposproject.org/node/304`). INGENIAS contributes providing the code of complete projects (see `http://ingenias.svn.sourceforge.net/viewvc/ingenias/trunk/IDK/iaf/tests/`). These developments are actually used as regression tests to verify that code generation capabilities of INGENIAS are not altered.

Here it is important to distinguish between research and industrial case studies. The first ones can be enough for initial validation stages, but authentic validation comes when the methodology meets the reality. For instance, Weyns (2010) introduces an architectural oriented method applied to design software for a transportation system developed between 2004 and 2007 together with Egemin company. This case study addresses the transport of loads within an industrial development, such as a warehouse, in a decentralized manner. The generic method is an adaptation of existing methods developed in the Software Engineering Institute. From the collaboration with Egemin, Weyns highlights three lessons: the importance of dedicating effort to design a system architecture that meets functional and non-functional requirements of the domain problem, the relevance of documentation, and the integration of agent methods with a mainstream SE approach.

In this line of thinking, the Dagstuhl Seminar on Engineering MAS (Dix *et al.*, 2012) gathered researchers from the academy and industry to share lessons on engineering MAS. Organizers took the opportunity to identify what was missing in agent technology that prevented industrial uptake. There were many interesting suggestions along the week, but three of them are specifically relevant to this paper: understanding the agent-oriented software lifecycle, the increasing importance of simulations, and the utter irrelevance of the technology for the client. Today there is a lack of understanding of the complete software lifecycle of agent-oriented software. Not knowing it, means we find hard to explain how, for instance, a MAS-oriented product can be maintained along the years after being delivered to the client. It may not be impossible to maintain a MAS software product. It may be too expensive, though, and we need to know in advance. Another issue is the increasing importance of simulations in different domains. Examples in the aeronautics and aerospace domains were introduced that helped to debug the MAS at the same time it served to communicate better the industrial staff what was being done. These simulations were used to validate the specification and the later design of the system. The third lesson is a killing one: clients do not mind whether the method is agent-oriented or not as long as it works as expected. Clients are not going to buy anything just because it was made with an AOSE methodology; they will buy it because the trade-off between quality and price is acceptable.

Our own experience with industry tells there is also a need to compare your methods with other existing ones. It is important to evaluate the benefits against the drawbacks. A convincing argument is the development costs. Our experimental attempt to prove AOSE development methods were more effective was published in Gómez-Sanz *et al.* (2005). There we try to foresee the costs of an agent-oriented development versus an object-oriented one. The way we did was to use experimental data using real work done for Telefonica I + D. The balance was in favor of agents but, as the paper says, more experimental data is required and larger developments needed in order to confirm this preliminary result.

There is much to learn from SE and many aspects to improve in AOSE methodologies. Looking at the software lifecycle as defined in IEEE (1990), we have still to answer too many questions about how each phase has to be addressed when using agents. Looking for advise, there is a work from Lethbridge (2000), based on a survey to company staff and other volunteers, that studies which subjects are important to software practitioners. The survey is interesting because it actually serves to identify what AOSE methodologies should improve first. Participants were asked about 75 topics selected from university curricula and previous drafts from SWEBOK (Abran *et al.*, 2004). A 40% of participants worked for industry producing software. It turns out software architectures are one of the top four topics in this survey of 25 most important and 25 less important topics. Other top topics are data structures, software design and patterns, and specific programming languages. This survey identifies other relevant aspects important for the participants: requirements gathering and analysis methods, analysis and design methods, testing/verification/quality assurance methods, project management, configuration and release management, Human–Computer Interaction, and databases.

There are papers in agent research dealing with some of the above-mentioned aspects. This paper has pointed out a few in requirements and design methods, for instance. Nevertheless, we still need to gain a more development-oriented vision in line with mainstream SE. Perhaps the way to go is focusing on the rest of topics this survey identifies and determining how agent technology can contribute to them.

5 Conclusions

This paper has made an assessment of the qualities of AOSE methodologies, focusing on their proximity to industrial practices. The basic assumption we have made is that industry is driven by SE practices. Hence, the evaluation has attended to the elements SE considers relevant in a methodology. To make this evaluation, this paper has elaborated a definition of an *AOSE methodology* using as basis reference works from SE.

The definition of AOSE methodology has not been sufficiently exploited and could lead to new insights about our methodologies, for instance studying in more detail how each phase is addressed by each methodology. This paper has focused on the vocabulary and notation, the phases, and tools. As a result, there are four tables representing the contributions of 13 lines of research. These tables show how some works focus on a few development phases while others try to cover the whole software lifecycle. There is an evident difference in the scope of the methods, though this distinction does not mean some are better.

The evaluation of the AOSE works tells most effort has been devoted to requirements, design, and implementation phases. Nevertheless, advances are still needed in all phases of the software lifecycle, not just in a few. Perhaps we have ignored other phases because we did not have the kind of problems deserving them. If this is the case, the future of AOSE methodologies will depend on having case studies more ambitious than our current ones, specially those that imply several years of work and involve many stakeholders and developers. Anyone can see this is as an expensive experimentation that requires an important funding, but it is the kind of problems SE was created to deal with.

Acknowledgments

This paper has been funded by the the project SociAAL (Social Ambient Assisted Living), supported by Spanish Ministry for Economy and Competitiveness, with grant TIN2011-28335-C02-01 and by the Project for Innovation and Improvement of Teaching (Proyectos de Innovación y Mejora de la Calidad Docente 2010) number 127 from the UCM.

References

Abran, A., Bourque, P., Dupuis, R., Moore, J. W. & Tripp, L. L. (eds) 2004. *Guide to the Software Engineering Body of Knowledge—SWEBOK*, IEEE Press. `http://www.swebok.org`.

Aldewereld, H. & Dignum, V. 2010. OperettA: organization-oriented development environment. In *LADS*, Dastani, M., Fallah-Seghrouchni, A. E., Hübner, J., Leite, J. (eds), LNCS, **6822**, 1–18. Springer.

Argente, E., Beydoun, G., Fuentes-Fernández, R., Henderson-Sellers, B. & Low, G. 2011a. Modelling with Agents. In *AOSE X*, Gleizes, M. P., Gómez-Sanz, J. (eds), LNCS, **6038**, 157–168. Springer.

Argente, E., Botti, V. J. & Julián, V. 2011b. GORMAS: an organizational-oriented methodological guideline for Open MAS. In *AOSE X*, Gleizes, M. P., Gómez-Sanz, J. J. (eds), LNCS, **6038**, 32–47. Springer.

Argente, E., Julián, V. & Botti, V. J. 2008. MAS modeling based on organizations. In *AOSE IX*, Luck, M., Gómez-Sanz, J. (eds), LNCS, **5386**, 16–30. Springer.

Bernon, C., Camps, V., Gleizes, M. P. & Picard, G. 2005. Multi-agent systems: the ADELFE methodology. In *Agent-Oriented Methodologies*, Henderson-Sellers, B. & Giorgini, P. (eds), Idea Group Publishing, Ch. VII. 172–202.

Bertolini, D., Delpero, L., Mylopoulos, J., Novikau, A., Orler, A., Penserini, L., Perini, A., Susi, A. & Tomasi, B. 2006. A Tropos model-driven development environment. In *CEUR Workshop Proceedings*, Boudjlida, N., Cheng, D., Guelfi, N. (eds), **231**. CAiSE Forum. `CEUR-WS.org`.

Brinkkemper, S. 1996. Method engineering: engineering of information systems development methods and tools. *Information and Software Technology* **38**, 275–280.

Caire, G., Cossentino, M., Negri, A., Poggi, A. & Turci, P. 2004. Multi-agent systems implementation and testing. In *Proceedings of 4th International Symposium From Agent Theory to Agent Implementation (AT2AI-4).*

Cervenka, R., Greenwood, D. A. P. & Trencanský, I. 2006a. The AML approach to modeling autonomic systems. In *2006 International Conference on Autonomic and Autonomous Systems (ICAS '06).* IEEE Computer Society, 29.

Cervenka, R., Trencanský, I. & Calisti, M. 2006b. Modeling social aspects of multi-agent systems: the AML approach. In *AOSE VI*, Müller, J. P., Zambonelli, F. (eds), LNCS, **3950**, 28–39. Springer.

CMMI Product Team. 2002. *Capability Maturity Model Integration (CMMISM) v. 1.1. Tech. Rep.*, Carnegie Mellon, Software Engineering Institute.

Cossentino, M. 2005. From requirements to code with the PASSI methodology. In *Agent-Oriented Methodologies*, Henderson-Sellers, B. & Giorgini, P. (eds), Idea Group Publishing, Ch. IV. 79–106.

Cossentino, M., Gleizes, M. P., Molesini, A. & Omicini, A. 2011. Processes engineering and AOSE. In *AOSE X*, Gleizes, M. P., Gomez-Sanz, J. (eds), LNCS, **6038**, 191–212. Springer.

Cossentino, M., Sabatucci, L. & 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)*, 539–544.

DeLoach, S. A. & Kumar, M. 2005. Multi-agent systems engineering: an overview and case study. In *Agent-Oriented Methodologies*, Henderson-Sellers, B. & Giorgini, P. (eds), Idea Group Publishing, Ch. XI. 317–340.

DeLoach, S. A. & Wood, M. F. 2000. Developing multiagent systems with agentTool. In *ATAL*, Castelfranchi, C., Lespérance, Y. (eds), LNCS, **1986**, 46–60. Springer.

Dix, J., Hindriks, K. V., Logan, B. & Wobcke, W. 2012. Dagstuhl Seminar on Engineering Multi-Agent Systems. `http://www.dagstuhl.de/en/program/calendar/semhp/?semnr=12342`.

El Fallah-Seghrouchni, A., Gómez-Sanz, J. & Singh, M. 2011. Formal methods in agent-oriented software engineering. In *AOSE X*, Gleizes, M. P., Gómez-Sanz, J. (eds), LNCS, **6038**, 213–228. Springer.

Ferber, J. & Gutknecht, O. 1998. A meta-model for the analysis and design of organizations in multi-agent systems. In *ICMAS*, Demazeau, Y. (ed.), IEEE Computer Society, 128–135.

Fuentes, R., Gómez-Sanz, J. J. & Pavón, J. 2006. Requirements elicitation for agent-based applications. In *AOSE VI*, Müller, J. P., Zambonelli, F. (eds), LNCS, **3950**, 40–53. Springer.

Fuxman, A., Mylopoulos, J., Pistore, M. & Traverso, P. 2001. Model checking early requirements specifications in Tropos. In *Proceedings of 5th IEEE International Symposium on Requirements Engineering (RE 2001),* 174–181.

García, E., Argente, E. & Giret, A. 2010. EMFGormas: a CASE tool for developing service-oriented open MAS. In *Proceedings of 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, W., Kaminka, G. A., Lespérance, Y., Luck, M., Sen, S. (eds), **1**. IFAAMAS, 1623–1624.

García-Magariño, I., Gómez-Sanz, J. J. & Pérez-Agüera, J. R. 2008. A multi-agent based implementation of a Delphi process. In *Proceedings of 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, L., Parkes, D. C., Müller, J. P., Parsons, S. (eds), **3**. IFAAMAS, 1543–1546.

García-Ojeda, J. C., DeLoach, S. A. & Robby. 2009. agentTool III: from process definition to code generation. In *Proceedings of 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Sierra, C., Castelfranchi, C., Decker, K. S., Sichman, J. S. (eds), **2**. IFAAMAS, 1393–1394.

Garijo, F. J., Gómez-Sanz, J. J. & Massonet, P. 2005. The MESSAGE methodology for agent-oriented analysis and design. In *Agent-Oriented Methodologies*, Henderson-Sellers, B. & Giorgini, P. (eds), Idea Group Publishing, Ch. VIII. 203–234.

Georgeff, M. P. 2009. The gap between software engineering and multi-agent systems: bridging the divide. *IJAOSE* **3**, 391–396.

Giorgini, P., Kolp, M., Mylopoulos, J. & Castro, J. 2005a. Tropos: a requirements-driven methodology for agent-oriented software. In *Agent-Oriented Methodologies*, Henderson-Sellers, B. & Giorgini, P. (eds), Idea Group Publishing, Ch. II. 20–45.

Giorgini, P., Mylopoulos, J. & Sebastiani, R. 2005b. Goal-oriented requirements analysis and reasoning in the tropos methodology. *Engineering Applications of Artificial Intelligence* **18**, 159–171.

Gómez-Sanz, J. J. 2007. *INGENIAS Agent Framework: Development Guide*, Universidad Complutense de Madrid. `http://sourceforge.net/projects/ingenias/files/INGENIAS%20Development%20Kit/Aranjuez/IAFDevelopmentGuide.pdf/download`.

Gómez-Sanz, J. J., Bota-Blaya, J. A., Serrano, E. & Pavón, J 2009. Testing and debugging of MAS interactions with INGENIAS. In *AOSE IX*, Luck, M., Gómez-Sanz, J. (eds), LNCS, **5386**, 199–212. Springer.

Gómez-Sanz, J. J., Fernández, C. R. & Arroyo, J. 2010. Model driven development and simulations with the INGENIAS agent framework. *Simulation Modelling Practice and Theory* **18**, 1468–1482.

Gómez-Sanz, J. J., Fuentes, R., Pavón, J. & García-Magariño, I. 2008. INGENIAS development kit: a visual multi-agent system development environment. In *Proceedings of 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Demo Proceedings*. IFAAMAS, 1675–1676.

Gómez-Sanz, J. J. & Pavón, J. 2002. Meta-modelling in Agent Oriented Software Engineering. In *IBERAMIA*, Garijo, F. J., Santos, J. C. R., Toro, M. (eds), LNCS, **2527**, 606–615. Springer.

Gómez-Sanz, J. J. & Pavón, J. 2005. Implementing multi-agent systems organizations with INGENIAS. In *PROMAS*, Bordini, R. H., Dastani, M., Dix, J., Fallah-Seghrouchni, A. E. (eds), LNCS, **3862**, 236–251. Springer.

Gómez-Sanz, J. J., Pavón, J. & Garijo, F. J. 2005. Estimating costs for Agent Oriented Software. In *AOSE*, Müller, J. P., Zambonelli, F. (eds), LNCS, **3950**, 218–230. Springer.

Gorodetsky, V., Karsaev, O., Samoilov, V. & Konushy, V. 2008. Support for analysis, design, and implementation stages with MASDK. In *AOSE IX*, Luck, M., Gómez-Sanz, J. (eds), LNCS, **5386**, 272–287. Springer.

Graja, Z., Regayeg, A. & Kacem, A. H. 2010. ForMAAD: towards a model driven approach for agent based application design. In *AOSE XI*, Weyns, D., Gleizes, M. P. (eds), LNCS, **6788**, 148–164. Springer.

Hahn, C. & Fischer, K. 2008. The formal semantics of the domain specific modeling language for multiagent systems. In *AOSE IX*, Luck, M., Gómez-Sanz, J. (eds), LNCS, **5386**, 145–158. Springer.

Hahn, C., Zinnikus, I., Warwas, S. & Fischer, K. 2009. From agent interaction protocols to executable code: a model-driven approach. In *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Sierra, C., Castelfranchi, C., Decker, K. S., Sichman, J. S. (eds), **2**. IFAAMAS, 1199–1200.

Henderson-Sellers, B. & Giorgini, P. (eds) 2005. *Agent-Oriented Methodologies*, Idea Group.

IEEE. 1990. IEEE standard glossary of software engineering terminology. *IEEE Std* **610**, 121990.

Iglesias, C. A., Garijo, M., Centeno-González, J. & Velasco, J. R. 1997. Analysis and design of multiagent systems using MAS-Common KADS. In *ATAL*, Singh, M. P., Rao, A. S., Wooldridge, M. (eds), LNCS, **1365**, 313–327. Springer.

Kacem, A. H., Regayeg, A. & Jmaiel, M. 2007. ForMAAD: a formal method for agent-based application design. *Web Intelligence and Agent Systems* **5**, 435–454.

Kelly, S., Lyytinen, K. & Rossi, M. 1996. MetaEdit+: a fully configurable multi-user and multi-tool CASE and CAME environment. In *CAiSE*, Constantopoulos, P., Mylopoulos, J., Vassiliou, Y. (eds), LNCS, **1080**, 1–21. Springer.

Kendall, E. A. 1999. Role modeling for agent system analysis, design, and implementation. In *Proceedings of 3rd International Symposium on Mobile Agents/Agent Systems and Applications (ASA/MA 1999).* IEEE Computer Society, 204–218.

Lethbridge, T. C. 2000. What knowledge is important to a software professional? *Computer* **33**, 44–50.

Luck, M., McBurney, P., Shehory, O. & Willmott, S. 2005. Agent technology: computing as interaction (a roadmap for agent based computing). *AgentLink*, `http://www.agentlink.org/roadmap`.

Luck, M. & Padgham, L. (eds) 2008. Agent-Oriented Software Engineering VIII, LNCS, **4951**. Springer.

Massonet, P., Deville, Y. & Nève, C. 2002. From AOSE methodology to agent implementation. In *Proceedings of 1st International Joint Conference on Autonomous Agents & Multiagent Systems (AAMAS 2002)*. ACM, 27–34.

Morandini, M., Nguyen, D. C., Perini, A., Siena, A. & Susi, A. 2008. Tool-supported development with Tropos: The Conference Management System Case Study. In *AOSE VIII*, Luck, M., Padgham, L. (eds), LNCS, **4951**, 182–196. Springer.

Naur, P. & Randell, B. (eds) 1968. *Software Engineering, report on a conference sponsored by the NATO SCience Committee*. NATO, Science Affairs Division from NATO.

Nguyen, C. D., Perini, A. & Tonella, P. 2010. Goal-oriented testing for MASs. *International Journal of Agent Oriented Software Engineering* **4**, 79–109.

Nunes, I., Cirilo, E., de Lucena, C., Sudeikat, J., Hahn, C. & Gómez-Sanz, J. 2011. A survey on the implementation of agent oriented specifications. In *AOSE X*, Gleizes, M. P., Gomez-Sanz, J. (eds), LNCS, **6038**, 169–179. Springer.

Object Management Group. 2008. Software and Systems Process Engineering Metamodel Specification (SPEM) 2.0.

Padgham, L., Thangarajah, J. & Winikoff, M. 2008. The Prometheus design tool—A Conference Management System Case Study. In *AOSE VIII*, Luck, M., Padgham, L. (eds), LNCS, **4951**, 197–211. Springer.

Padgham, L. & Winikoff, M. 2004. *Developing Intelligent Agent Systems: A Practical Guide*, John Wiley & Sons.

Padgham, L. & Winikoff, M. 2005. Prometheus: a practical agent-oriented methodology. In *Agent-Oriented Methodologies*, Henderson-Sellers, B. & Giorgini, P. (eds), Idea Group Publishing, Ch. V. 107–135.

Parnas, D. L. 1999. Software engineering programs are not computer science programs. *IEEE Software* **16**, 19–30.

Paulk, M., Curtis, B., Chrissis, M. & Weber, C. 1993. Capability maturity model v. 1.1. *Software, IEEE* **10**, 18–27.

Pavón, J. & Gómez-Sanz, J. J. 2003. Agent oriented software engineering with INGENIAS. In *CEEMAS*, Mark, V., Müller, J. P., Pechoucek, M. (eds), LNCS, **2691**, 394–403. Springer.

Pavón, J., Gómez-Sanz, J. J. & Fuentes, R. 2005. The INGENIAS methodology and tools. In *Agent-Oriented Methodologies*, Henderson-Sellers, B. & Giorgini, P. (eds), Idea Group Publishing, Ch. IX. 236–276.

Pavón, J., Gómez-Sanz, J. J. & Fuentes, R. 2006. Model driven development of multi-agent systems. In *ECMDA-FA*, Rensink, A.,Warmer, J. (eds), LNCS, **4066**, 284–298. Springer.

Regayeg, A., Kacem, A. H. & Jmaiel, M. 2004. Specification and design of multi-agent applications using Temporal Z. In *PRIMA*, Barley, M., Kasabov, N. K. (eds), LNCS, **3371**, 228–242. Springer.

Rougemaille, S., Arcangeli, J. P., Gleizes, M. P. & Migeon, F. 2008. ADELFE design, AMAS-ML in action. In *ESAW*, Artikis, A., Picard, G., Vercouter, L. (eds), LNCS, **5485**, 105–120. Springer.

Sabatucci, L., Cossentino, M. & Gaglio, S. 2006. Building agents with agents and patterns. In *WOA*, *CEUR Workshop Proceedings*, Paoli, F. D., Stefano, A. D., Omicini, A., Santoro, C. (eds), **204**. `CEUR-WS.org`.

Seidita, V., Cossentino, M. & Gaglio, S. 2009. Using and extending the SPEM specifications to represent agent oriented methodologies. In *AOSE IX*, Luck, M., Gómez-Sanz, J. (eds), LNCS, **5386**, 46–59. Springer.

Sierra, C., Castelfranchi, C., Decker, K. S. & Sichman, J. S. (eds) 2009. *8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, **2**. IFAAMAS.

Spanoudakis, N. I. & Moraitis, P. 2008. The Agent Modeling Language (AMOLA). In *AIMSA*, Dochev, D., Pistore, M., Traverso, P. (eds), LNCS, **5253**, 32–44. Springer.

Spanoudakis, N. I. & Moraitis, P. 2010. Modular JADE agents design and implementation using ASEME. In *IAT*, Huang, J. X., Ghorbani, A. A., Hacid, M.-S. & Yamaguchi, T. (eds), IEEE Computer Society Press, 221–228.

Spanoudakis, N. I. & Moraitis, P. 2011. Using ASEME methodology for model-driven agent systems development. In *AOSE XI*, Weyns, D., Gleizes, M. P. (eds), LNCS, **6788**, 106–127. Springer.

Sturm, A. & Shehory, O. 2003. A framework for evaluating agent-oriented methodologies. In *AOIS*, Giorgini, P., Henderson-Sellers, B., Winikoff, M. (eds), LNCS, **3030**, 94–109. Springer.

Sun, H., Thangarajah, J. & Padgham, L. 2010. Eclipse-based Prometheus Design Tool. In *Proceedings of 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, W., Kaminka, G. A., Lespérance, Y., Luck, M., Sen, S. (eds), **1**. IFAAMAS, 1769–1770.

van Putten, B. J., Dignum, V., Sierhuis, M. & Wolfe, S. R. 2009. OperA and Brahms: a symphony? In *AOSE IX*, Luck, M., Gómez-Sanz, J. (eds), LNCS, **5386**, 257–271. Springer.

Warwas, S. & Hahn, C. 2009. The DSML4MAS development environment. In *Proceedings of 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Sierra, C., Castelfranchi, C., Decker, K. S., Sichman, J. S. (eds), **2**. IFAAMAS, 1379–1380.

Weyns, D. 2010. *Architecture-Based Design of Multi-Agent Systems*, Springer-Verlag.

Yu, E. S. K. 1997. Towards modeling and reasoning support for early-phase requirements engineering. In *3rd {IEEE} International Symposium on Requirements Engineering*. IEEE Computer Society, 226–235.

Zambonelli, F., Jennings, N. R. & Wooldridge, M. 2005. Multi-agent systems as computational organizations: the Gaia methodology. In *Agent-Oriented Methodologies*, Henderson-Sellers, B. & Giorgini, P. (eds), Idea Group Publishing, Ch. VI. 136–171.

Zhang, Z., Thangarajah, J. & Padgham, L. 2011. Automated testing for intelligent agent systems. In *AOSE X*, Gleizes, M.-P., Gómez-Sanz, J. (eds), LNCS, **6038**, 66–79. Springer.