PROCESS MODELS FOR AGENT-BASED DEVELOPMENT

Luca Cernuzzi[1,2], Massimo Cossentino[3], Franco Zambonelli[1]

1) DISMI – Università di Modena e Reggio Emilia, Italy
2) Universidad Catolica de Asunciòn, Paraguay
3) Istituto Calcolo Alte Prestazioni – CNR Palermo, Italy
e-mail: cernuzzi.luca@unimore.it, cossentino@pa.icar.cnr.it, franco.zambonelli@unimore.it

ABSTRACT

A great deal of researches in the area of agent-oriented software engineering focuses on proposing methodologies for agent systems, i.e., on identifying the guidelines to drive the various phases of agent-based software development and the abstractions to be exploited in these phases. However, very little attention has been paid so far to the basic issue of engineering the process subjacent the development activity, i.e., of disciplining the execution of the different phases involved in the software lifecycle. In this paper, we focus on process models for software development and put these in relations with current researches in agent-oriented software engineering. First, we introduce the key concepts and issues related to software processes and present the various process models currently adopted in mainstream software engineering. Then, we survey the characteristics of a number of agent-oriented methodologies, as they pertain to software processes. In particular, for each methodology, we analyze which process model it (often implicitly) underlies and which phases of the process are covered by it, thus enabling us to identify some key limitations of currently methodology-centered researches. On this base, we eventually identify and analyze several open issues in the area of process models for agent-based development, calling for further researches and experiences.

KEYWORDS: *Agent-based Computing, Software Engineering, Methodologies, Process Models.*

1. INTRODUCTION

Agents and multiagent systems, other than a technology, represent a brand new paradigm for software development (Jennings 2001; Zambonelli et al, 2003). When adopting agents as the basic conceptual components of software systems, software has to be conceived in terms of autonomous task-oriented entities, interacting with each other in a high-level way (e.g., via co-operation., coordination of activities, negotiations), leading to possibly very articulated organizations (e.g, teams, coalitions, markets, swarms). This sharply contrasts with the traditional idea of software as made up of functional-oriented entities (i.e., objects or components) interacting in a client-server way in the context of static architectures, and calls from a brand new set of tools to support software development.

In this context, *agent-oriented software engineering* (from now on AOSE) (Ciancarini and Wooldridge, 2001; Zambonelli and Omicini, 2004) has recently emerged as the discipline devoted to the engineering of complex software systems based on the multiagent systems paradigm. Researches in the area of AOSE include the identification and development of both conceptual tools (e.g., formal modeling and notational tools) and practical tools (e.g., agent-based infrastructures and case tools) to support software engineers and programmers in the analysis, design and development of multiagent systems. Among the others, a great deal of efforts in the AOSE area focuses

on the definition of *methodologies* to guide the process of developing multiagent systems.

AOSE methodologies, as they have been proposed so far, mainly try to suggest a clean and disciplined approach to analyze, design and develop multiagent systems, using specific methods and techniques. To this end, AOSE methodologies, as well as non-AOSE ones (Lahlouhi and Sahnoun, 2002), typically start from a meta-model, identifying the basic abstractions to be exploited in development (e.g., agents, roles, environment, organizational structures). On this base, they exploit and organize these abstractions so as to define guidelines on how to proceed in the analysis, design, and development, and on what output to produce at each stage. Unfortunately, this is far to be enough for practical software development. In particular, the adoption of a methodology per se is of little use without a clear understanding of the *software development process model* that should underlie it.

In general, one of the very goals of software engineering researches is increasing the quality and the reliability of software, as well as improving the cost-effectiveness of software development. In this context, the basic understanding is that there is a direct correlation between the quality of the software being developed and the quality of the software development process (Fuggetta, 2000), i.e., the overall execution and coordination of the different phases that lead to the desired product. While this consideration is valid for any human product, in the case of software it assumes even greater emphasis due to factors such as intangibility of the product, high-instability of requirements, and – in the case of multiagent systems – dynamics of operational environments and of the multiagent systems themselves.

Accordingly, in the development of software systems and of multiagent systems, the identification of a suitable methodology cannot abstract from the identification of a specific model for the software development process (Boehm, 1988). Such model should define how the different phases of software development should be organized and coordinated with each other, which activities engineers and developers have to undertake in each stage and when, which technologies and artifacts may be used for those activities, which products have to be expected for each stage, and which resources need to be involved in the phases of software production process. In other word, the software process model should guide all the production effort and complement the guidelines identified by a specific methodology.

In recognition of the fact that current researches on AOSE methodologies mostly disregard any considerations about process models, the contribution of this paper is to go into details about software process models and analyze a few key questions that – in our opinion – will be of a key importance of the future of the AOSE discipline (Zambonelli and Omicini, 2004). In particular:

- Does the choice of a specific software process model impact on the activity of agent-oriented software development? How can past and current experiences in the area of software process models impact researches in the area of agent-oriented software engineering?
- Which methodologies, either implicitly or explicitly, adopt which process model? Can a specific methodology be applied in the context of a specific process model?
- What further research is needed in this area? Which research trends and open issues are relevant and may be envisioned in AOSE arena from the process model point of view? Are traditional software process models adequate for multiagent systems?

According to the above key questions, and for the sake of elaborating on them, the remainder of this paper is organized as follows. Section 2 introduces the concept of

software process models and discusses the traditional and emerging trends in the area, also in relationship with agent-based computing. Section 3 presents a survey of some AOSE methodologies proposed in the literature, with a specific focus on the characteristics related to the process model perspective. Section 4 identifies some open issues and promising research directions in the area of process models for agent-oriented software engineering. Section 5 concludes.

2. SOFTWARE PROCESS MODELS

In this section, we introduce the basic definitions and concepts related to software development process models, survey the main process models proposed so far in the area, and put this in relation with agent-based software development and agent-oriented methodologies.

2.1 Processes vs. Methodologies

Process and methodology are often used with contradictory meanings, sometimes they are seen as synonymous other times as different or complementary terms. In order to prevent confusions and misunderstandings, we introduce a minimal glossary.

In this paper we will use the terms process, software process, software process model, method, and methodology in accord to the following meanings:

- **(Development) Process**. Generally speaking, a development process (or simply process) is an ordered set of steps that involve all those activities, constraints and resources required to produce a specific desired output (e.g., a physical artifact) satisfying a set of input requirements. Typically, a process is composed by different stages/phases put in relation with each other. Each stage/phase of a process identify a portion of work (more properly called work definition) to be done in the context of the process, the resources to be exploited to that purpose and the constraints to be obeyed in the execution of the phase. Case by case, the work in a phase can be very small (in complexity, spent time and required effort) or more demanding (thus arriving to include significant portions of the whole process). Phases are usually composed by a set of activities that may be, in turn, be conceived in terms of smaller atomic units of work (steps).
- **Software (Development) Process**. Looking more specifically at the software context, Fuggetta (2000) proposes an interesting definition of software development process (or simply software process): "*the coherent set of policies, organizational structures, technologies, procedures, and artifacts that are needed to conceive, develop, deploy, and maintain (evolve) a software product*". Starting from this definition, we can identify that also software processes can be (and are typically) composed by a set of stages, each specifying which phases/activities should be carried on and which roles (i.e.: client, analyst, software architect, programmers, etc.) and resources are to be involved in them. A process – whether involving software or not – will also prescribe in which sequence and according to which time scheduling the phases and activities will be executed and if iterations should be done or feedbacks provided to previous stages; this often relates to some common process models (see next definition). Commonly, the stages of a software process aim at producing/refining a number of conceptual and digital artifacts (text documents, diagrams, code fragments, modules, software agents…) and in so doing they often refer to some kind of template (text documents) or modeling language (diagrams). However, unlike

traditional "development" processes, software processes should also take into account the fact that the product should not only be developed but also: *(i)* "conceived", often relying on unstable or incomplete requirements; *(ii)* deployed, i.e., put to work in an operational environment; *(iii)* maintained and evolved, depending on novel requirements or changes in the operational environments.

- **Software (Development) Process Model**. A software (development) process model prescribes around which phases a process should be organized (and possibly but not necessarily which activities should be executed in some of the phases), in which order such phases should be executed, and when iterations and coordination (if any) between the work of the different phases should occur. In other words, a process model defines a skeleton, a template, around which to organize and detail an actual process. A software development process model (that, from now on, we will simply indicate as a "process model") does not take care of fine-grained work definitions, guidelines, modeling style for artifacts, as these can change and be adapted from case to case. This is one of the most important aspects of process models, and it will be the subject of a detailed analysis in the following sub-section, where we will describe the most common process models and then, in another section and will relate them to some of the most common agent-oriented methodologies.
- **Method**. A method prescribes a way of performing some kind of activity within a process, in order to properly produce a specific output (i.e., an artifact or a document) starting from a specific input (again, an artifact or a document). Any phases of a process, to be successfully applicable, should be complemented by some methodological guidelines (including the identification of the techniques and tools to be used, and the definition of how artifacts have be produced) that could help the involved stakeholders in accomplishing their work according to some defined best practices. For instance, in the object-oriented context, while performing the identification of components of the requirements analysis model, a guideline could suggest of starting from textual scenarios and it could describe the use of the Abbot technique (Abbot, 1983) (which prescribes that proper nouns become objects, improper nouns become classes, doing verbs become methods and so on).
- **Methodology**. A methodology is a collection of methods covering and connecting different stages in a process. The purpose of a methodology is to prescribe a certain coherent approach to solving a problem in the context of a software process by preselecting and putting in relation a number of methods (Ghezzi et al., 1991).

Based on the above definitions, and comparing software processes and methodologies, we can find some common elements in their scope. Both are focusing on *what* we have to do in the different activities needed to construct a software system. However, while the software development process is more centered on the global process including all the stages, their order and time scheduling, the methodology focuses more directly on the specific techniques to be used and artifacts to be produced (i.e. in the methods to be adopted). In this sense, we could say that methodologies focus more explicitly on *how* to perform the activity or tasks in some specific stages of the process, while processes may also cover more general management aspects, e.g., basic questions about *who* and *when*, and *how much*).

2.2. Process Models

Let us now analyze the most relevant process models proposed so far in the literature and applied in industry.

The *waterfall* process model has been for many years the emblem of software engineering; it firstly appeared in the late 1950s but became popular in the 1970s. It is structured as a cascade of phases, where the output of one phase constitutes the input of the next one. Each phase, in turn, is structured as a set of activities that might be executed by different people concurrently. There are many variants of the waterfall model (e.g., Figure 1) but, despite their differences, all existing waterfall processes look alike and share the same underlying philosophy: they prescribe a sequential, linear flow among phases, adopting standards on the way the outputs (deliverables) of each stage must be produced, and sometimes prescribing methods to obtain the desired outputs. Feedbacks between different phases in the process may be required, forcing to step back by re-executing and tuning the activities of previous phases whenever the following phases outline problems.

One of the key advantages of the waterfall model is that it clearly identifies a number of relevant phases that a process of software development should pass through. These phases – although differently organized and with some small adaptation in scope and importance – are likely to appear in any other process models and include: *(i)* requirements elicitation, to collect the requirements for the system to be; *(ii)* requirements analysis, to organize the requirements into a set of functionalities to be provided by the system to be; *(iii)* design of the architecture of the system to be and of its components; *(iv)* coding and implementation of the system; *(v)* verification of the behavior of the system and of its components; *(vi)* deployment of the system into an operation environment.

Needless to say, multiagent systems development too should be involved in these phases, although sometimes with a different flavor than in traditional software development. Requirements analysis for an agent system typically translates into a set of "goals" to be pursued by the agents of the systems, rather than in a set of functionalities to be provided by, e.g., objects. Coding and implementation may not reduce in writing code fragments, but also in integrating in agents a set of world models, ontologies, communicative capabilities, and in shaping in a proper way the operational environment. Verification and testing may often imply a number of simulations to test the emergent behavior of the system, its capability to reach/approach a specific goal, and – in the case of open systems – the capability of its component agents to fruitfully interact and negotiate with agents in the external world. Deployment often implies putting the system at work in an existing multiagent system ecology, and at verifying the impact of the system in such ecology (Zambonelli and Omicini, 2004).

The waterfall model played an important role in the history of software engineering also because it firstly prescribed some indeed needed discipline in the software development process. However, the waterfall model accomplishes such discipline through rigidity of the model. Actually it is very hard to really use a prescriptive cascade and linear model without considering feedbacks on earlier stages (feedbacks that in the waterfall model are intrinsically costly) as primary components of a process. Unless a system has very stable and clear requirements, and is conceived to be deployed in a closed and static operational environment, more flexibility and adaptability in the process is necessary. For instance, the vast majority of agent applications focus on open multiagent systems that are based on a great number of agents whose interactions may produce an emergent behavior and distributed intelligence, and that are immersed in a

dynamic operational environment. Thus, the main limit of the waterfall model is that it neither anticipates changes in requirements, nor plans software evolution that now proves vital in many circumstances.

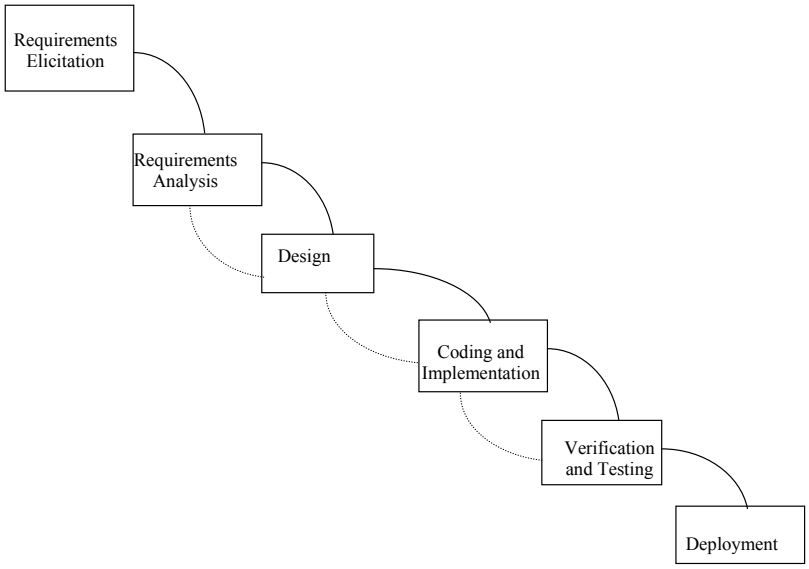

Figure 1. The Waterfall model (with some feedback)

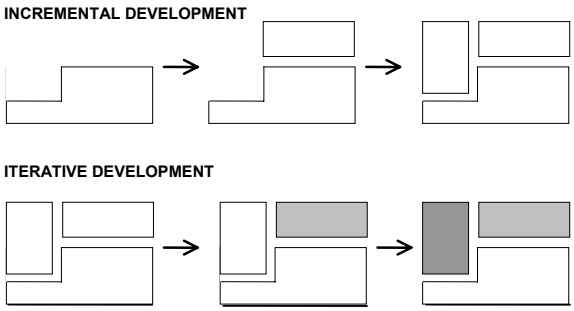

Figure 2. The Evolutionary (Incremental and Iterative) approach

The need for flexible approaches to process models, also called *evolutionary or incremental approaches*, has been widely acknowledged in the literature. An incremental approach consists of a stepwise development, where parts of some stages are postponed in order to produce some useful set of functions earlier in the development of the project. Boehm (1988) defines evolutionary process model ass a "models whose stages consist of expanding increments of an operational software product, with the direction of evolution being determined by operational experience". In other words, in an evolutionary approach, the traditional phases of the waterfall model are not intended to fully produce a complete systems, but rather to be executed multiple times in a quick way, so as to produce multiple version of the product and, on the basis of the feedback received, continue at working on the product, either by incrementally adding parts to it or by refining already implemented ones (Figure 2).

The strategy behind evolutionary process models may be simply synthesized as follows (Gilb, 1988): *(i)* design, code, and deploy something to the real user; *(ii)* measure the added value to the user in all critical dimensions; and *(iii)* adjust the

objectives and the design and implementation of the system based on observed realities; *(iv)* repeat as needed. However, it is important to observe that this pragmatic strategy has to be carried out in a structured and disciplined fashion. Clearly, since the iterative process of modeling at different levels a system may be resource-consuming, and the adopted solution often consists in reusing of as much as possible of the existing.

A number of specific models may be accommodated under the evolutionary approach (e.g., incremental implementation model, incremental development and delivery model, evolutionary prototype model, etc.). Probably, the most recent model in this direction, and perhaps one of the most interesting, is the *Extreme Programming* (XP) process model (Beck, 1999) that is being increasingly used in projects with uncertain or changing requirements. XP is an example of "agile approach" (see the agile manifest at www.agilemanifesto.org) aimed at supporting changes and rapid feedbacks. Agile approaches have become enough popular in the last years, their philosophy can be resumed by their fundamental strategies reported in the agile manifesto, which prescribed to give consideration to: *(i)* individuals and interactions over processes and tools; *(ii)* working software over comprehensive documentation; *(iii)* customer collaboration over contract negotiation; *(iv)* responding to change over following a plan.

In evolutionary approaches, and specifically in XP process models, the implementation and evaluation of executable code have priority over a comprehensive documentation and a strict software process with well-defined activities and modeling artifacts. For this reasons, it appears like these approaches can be suited for those small-medium-size project calling for rapid delivery and founded on quite unstable requirements. As far as multiagent systems are concerned, we can for example imagine that the adoption of an evolutionary approach can suit the development of an agent-enabled portal in which agents can be incrementally deployed and improved with regard to their capability of supporting users in accessing services and information. However, we can hardly imagine that agents and multiagent systems delegated of some critical activities, or multiagent systems that to be deployed in a context where their execution may have some dramatic impact, can be effectively be developed using a bare evolutionary approach, i.e., an XP model as it is currently conceived.

Another interesting approach, starting from a totally different philosophy than incremental models, is the *transformation model* (Ghezzi et al., 1991). Unlike incremental models, which give priority to implementation, transformation models are deeply rooted in theoretical works on formal specifications. The software development may be seen as a sequence of steps that gradually transform a set of formal specification into an implementation. The process consists of two main phases (Figure 3): requirement analysis, providing formal requirements specification (a phase which is given a much greater importance in this model than in the waterfall or in incremental ones) and optimization, which includes a coding phase aimed at transforming (possibly in an automatic way) the formal specifications into executable code and at performance tuning of the resulting system. Clearly, before being transformed, specifications are to be carefully verified against the user's expectations.

Currently, the transformation approach is not widespread in industry, because it requires an extensive use of formal modeling and also the presence of effective tools to transform formal specifications into actual code. Therefore, one can consider it a sort of research-oriented approach with a few actual tools that support it; an interesting example in the object oriented arena is OASIS that is associated to the OO-Method (Letelier et al. 1998; Pastor et al., 1997). However, when considering multiagent systems, one should take into account that – due to the intrinsic complexity of agent architectures and to the dynamics of multiagent systems, formal methods will definitely

play an increasing and fundamental role in agent-based development, possibly exploiting a transformation-inspired use of formal specifications in the context of some sorts of evolutionary process model.

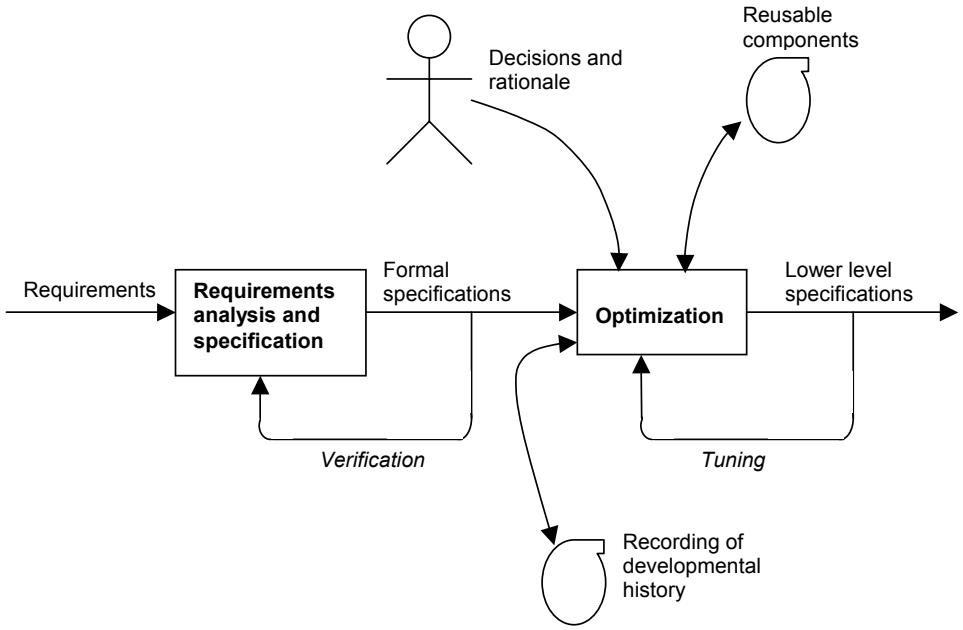

Figure 3. The Transformation model

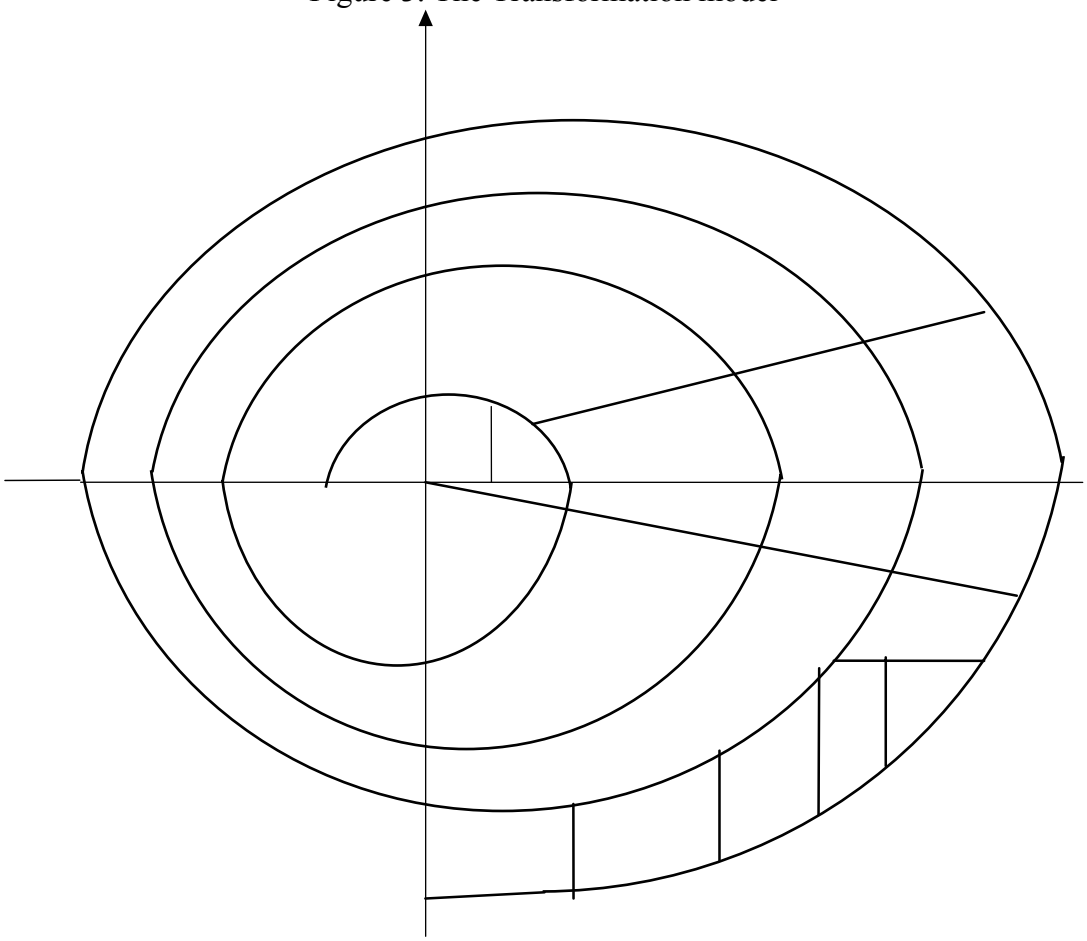

Figure 4. The Spiral model

The last process model we examine is the *spiral* one (Boehm, 1988), mainly focused on project risks and their management. In fact, its aim is to identify and eliminate high-risk problems in development by adopting a careful process design, rather than treating both trivial and severe problems uniformly. To this end, the main characteristic of the model is that it organizes the development process in a cyclic way (against the linear way of the waterfall). Each cycle of the spiral consists of four phases (Figure 4). Phase 1 (upper-left quadrant in Figure 4) aims at determining the objectives to be achieved in the cycle in terms of functional and quality levels, and at identifying possible alternatives and associated constraints. Phase 2 (upper-right quadrant) aims at evaluating the potential risks of the objectives and of the identified alternatives, eventually getting a decision on them. Phase 3 (bottom-right quadrant) aims at developing and verifying the next level product. Eventually, phase 4 (bottom-left quadrant) aims at reviewing the results of previous stages and at planning the next iteration of the spiral. As the spiral enlarge with the execution of different cycles, the risks involved in the project should more and more reduce, assuming that the more risky issues are faced in the inner cycles.

The spiral model is sometimes viewed as a kind of meta-model, because it can potentially accommodate any process development model. However, we prefer to consider a model by itself, well distinguished from the waterfall model (because of the explicit cyclic nature), from evolutionary approaches (because of the great importance given to planning and because the incremental development that the spiral model suggest is not aimed at producing deliverables but rather at eliminating development risks), and from the transformation one (whose cyclic nature mostly reduces at verifying the specifications). A representative example of a process based on the spiral model is the Rational Unified Process (Kruchten, 1998), which is iterative and incremental, use-case driven and centered on the architecture (attention on structural elements of the system and their relationships) of the system.

In general, the spiral model is very well suited for the development of all those complex software systems which involve high risks, and which require a careful planning to ensure that the final product will satisfy the specified (functional and non-functional) requirements. In the case of multiagent systems, one can think at several examples of mission critical applications that would fruitfully take advantage of the adoption of the spiral process model.

It is also worth mentioning that other process models aimed to specific areas of software production have been proposed. For instance, the *Statistical Process Control* model (Glaser and Haton 1996) has been conceived to maintain the process within pre-specified control limits and to guarantee in this way a given quality of the software product. In such model, a software development process is conceived as a set of tasks obtained through functional analysis, and refined in using day-to-day activity reports of software developers (Perry et al., 1994). The statistical control implies specifying a set of parameters and maintaining them – via statistical methods – within given limits. Proposals such as the above and other that can be found in the literature, being more oriented to management issues than development issues, will be no longer analyzed in this paper.

2.3. AOSE Methodologies and Process Models

Getting back to methodologies, an important issue to analyze is if a specific methodology can be exploited as a tool in the context of different process models or,

viceversa, if the adoption of a specific process model also suggests the exploitation of a methodology explicitly conceived for that process model.

Unfortunately, the relationship between process models and methodologies is someway nebulous. In general, a methodology should provide guidelines for the execution of some phases in a process model that should be independent from the specific way in which a process model prescribes to execute and coordinate such phases. However, in practice, it is hard to really obey to this. In fact, in most of the methodologies proposed so far (and, specifically, in most AOSE methodologies), the methodology itself has been conceived for adoption in the context of a specific process model. We do not claim this fact is always and necessarily bad: the commitment to a specific process model can make the methodology possibly less general but definitely more effective for practical application in the context of that specific process model. Nevertheless, and this is indeed bad, in several cases new AOSE methodologies gets proposed without explicitly relating them to some process models and, at the same time, being implicitly suitable only for a limited set of (or a single) process models.

As all of us know, the real process of software construction, if not controlled, can become be a chaotic effort with a low probability of reaching the desired goal within fixed limits of time and budget. Therefore, when an AOSE methodology is proposed with a lack of attention to the process model, this lack may strongly undermine the practical applicability of a methodology. As we have already analyzed, different process models differently drive the actions during the project enactment and are differently concerned with verification, control, comprehension, and improvement of the established activities. Moreover, while some well known and documented process models let easily capture good experiences and to transfer them to other persons, some others only aim at introducing a minimum level of control in the chaos of the software development thus allowing a high reactivity to very dynamic situations (they are adopted when facing frequently changing requirements). These differences in process models have a direct consequence: in order to have a good process and successfully complete the project, it is necessary to adopt either explicitly general methods and methodologies, or specifically suitable ones.

Another important aspect we should consider in our study of processes and methodologies is that a process is not something static that, once adopted in an industry or in a complex project, should never be changed. A process can evolve over time, together with the increased awareness of the involved people, towards a maturity level that ensures the process repeatability in terms of quality of the produced software, cost and time. This is a fundamental evaluation criterion for a company that wants to adopt the agent-based paradigm in its development process. In order to evaluate from this point of view existing AOSE processes and related methodologies, we could refer to the Process Capability Maturity Model (CMM) (Paulk et al., 1995). CMM proposes a classification of the process maturity in five different categories: *(i)* initial, i.e., the process is not effectively managed and its results are unpredictable because procedures and project plans are not formally defined; *(ii)* repeatable, i.e., the process management is sufficiently documented and some kind of quality assurance procedure is adopted but no formal process model exist; *(iii)* defined, i.e., the process is formally defined in all of its aspects and an organization can repeat it obtaining same quality results and also deal with its qualitative improvement; *(iv)* managed, i.e., the process is subject to the collection of quantitative data in order to evaluate its goodness and proceed to its improvement; *(v)* optimized, i.e., the process has been optimized on the basis of results analysis and its improvement becomes part of the overall project planning and budget. From the study of existing AOSE approaches to system development, it is quite easy to

evaluate them in terms of maturity: most of them are simply "initial" due to the lack of experience, and a few of them can classify as "repeatable". In any case, it is worth noting that the CMM model evaluates a process model in its completeness, thus including the methodological aspects that are so intensively studied in the AOSE context. In this way, the CMM could be used to evaluate not only the goodness of the adopted process model in a specific situation but also the appropriateness of each method, thus driving the evolution of the studied methodology towards a more mature one. On the contrary, evaluating the effectiveness methodology when it does not explicitly situates in the context of a process model, is of limited meaning.

3. A SURVEY OF AOSE METHODOLOGIES FORM THE PROCESS MODEL VIEWPOINT

In last few years a great deal of efforts has been spent in proposing AOSE methodologies to guide the development of multiagent systems. For full details on these, we forward the reader to some surveys in the area (Iglesias et al., 1999; Ciancarini and Wooldridge, 2000) as well as to the proceedings of workshop series: Agent-Oriented Software Engineering (AOSE), Agent Oriented Methodologies (AOM), Agent Oriented Information Systems (AOIS), Engineering Society in the Agent Word (ESAW), Software Engineering for Large-Scale Multi-Agent Systems (SELMAS).

However, as discussed in the previous section, a small attention has been devoted so far to analyze the fundamental issue of the related process models. For this reason, this section surveys a number of selected AOSE methodologies and analyze them with regard to those characteristics that are relevant from the software process viewpoint. Specifically, we would like to analyze the followings issues:

- Which process model do these AOSE methodologies, either explicitly or implicitly, assume?
- Which phases are covered by these methodologies? And, consequently, how suitable is a specific methodology to be exploited in the context of an actual process model?

In more detail, with regard to the first point, in the following we classify AOSE methodologies based on the class of process model (among the four presented in Section 2) it either assumes explicitly or that we consider to implicitly underlie the methodology. Therefore, we emphasize that whenever a methodology (i.e., the documents describing it) does not explicitly mention the process model it refers to, our classifying it as a methodology that assumes a specific process model had to rely on a personal analysis that may not necessarily reflect the original intentions of the proposers, and that sometimes had to sharpen shades.

With regard to the second point, it is very useful to specify for each methodology which stages or phases of the comprehensive software development process are covered. In effect, in some aspects the phase coverage is strongly related with the process. For example, if a methodology does not cover the coding and implementation, it can hardly be exploited in the context of extreme programming process models. As another example, if the methodology does not cover the requirement elicitation phase, its application in the context of a spiral process model or of a transformation model is strongly undermined. Although in the specialized literature there is not a general agreement about the naming and the roles of the various phases of software development (e.g., some propose a more specialized separation of the analysis or design phases trying to capture particular aspects like as conceptual, architectural, or detailed

design), the phases already identified in Section 2 when presenting the waterfall model are enough to the purposes of our analysis.

Clearly, as we had already the chance of discussing in Section 2, the possibility of applying a methodology in the context of a specific process model rather than others comes with drawback and advantages, and makes the methodology more suitable for specific types of application problems and less for others.

Figure 5 summarize the characteristics, to be analyzed and detailed in the following of this section, of several AOSE methodologies, i.e., the process model adopted and the phases of the software development process covered.

Phases → **Process Model** and **Methodology ↓**	Requirements Elicitation	Requirements Analysis	Design	Coding and Implementation	Verification & Testing	Deployment
Waterfall Like						
Gaia		X	X			
Roadmap	X (partially)	X	X			
Prometheus	X (partially)	X	X	X	X	
MaSE	X (partially)	X	X	X	X(partially)	
AOR		X	X	X		
Evolutionary and Incremental						
OPM/MAS	X	X	X			X
MASSIVE		X	X	X	X	X
Ingenias		X	X	X		
Tropos	X	X	X	X		
PASSI and Agile PASSI		X	X	X	X	X
Transformation						
DESIRE	X	X	X	X	X(partially)	
Spiral						
MAS-CommonKADs	X	X	X	X	X(partially)	X

Fig. 5. Methodologies Classification

Before continuing the presentation, we outline that in spite of a number of other papers trying to survey, classify, and evaluate, AOSE methodologies (Iglesias et al., 1999; Lahlouhi and Sahnoun, 2002; Cernuzzi and Rossi 2002), this paper is the first attempt specifically oriented to analyze the relations with the process model.

3.1. Waterfall-like

The methodologies that we feel should be considered as adopting a waterfall process model include: Gaia (Wooldridge et al., 2000; Zambonelli et al., 2003); Roadmap (Juan et al., 2003), as a consequence of the fact that is was firstly proposed as an extension of the original Gaia; Prometheus (Padgham and Winikoff, 2002); MASE (De Loach et al., 2001); AOR (Wagner, 2003). While for the first three methodologies their waterfall nature is rather clear, the last three, for different reasons, may be considered at the

borderline with evolutionary and incremental approach. In fact, they consider some kind of iteration inside some stages. Still, since they do not clearly stress on these interactions or on the incremental process, we prefer to classify them into the waterfall-like class.

Gaia

The Gaia methodology (Wooldridge et al., 2000) and its official extension (Zambonelli et al., 2003) focus on the use of organizational abstractions to drive the analysis and design of multi-agent systems. Gaia models both the macro (social) aspect and the micro (agent internals) aspect of a multiagent system, and devotes a specific effort to model the organizational structure and the organizational rules that govern the global behavior of the agents in the organization. This can make Gaia suitable for the development of multiagent systems which can interact in an open world with self-interested agents belonging to different stakeholders.

Gaia explicitly covers a limited number of phases in the design process, namely analysis and design (the latter included architectural and detailed design), Requirements elicitation, coding and implementation, verification and testing, and deployment, are not considered.

Gaia assumes in a rather explicit way a waterfall process model: analysis and design are considered as phases that should follow a (not defined) requirements elicitation phase, that should be performed in strict sequence, and for which methods for interactions between phases are not defined. In the analysis phase four basic models are produced: *(i)* the environmental model; *(ii)* a preliminary roles model; *(iii)* a preliminary interactions model; and *(iv)* a set of organizational rules. These models are used as input to the design phase that should define: *(i)* the overall organizational structure (i.e., the architecture of the system); *(ii)* completed preliminary roles and interactions models; *(iii)* an agent model; and *(iv)* a services model. The result of the design phase is assumed to be something that could be implemented in a technology neutral way.

Although we consider Gaia as one of the most promising approaches as far as analysis and design are concerned, the limited number of phases it covers and the strictly sequential approach may limit the adoption of Gaia, as it is, to systems with very stable requirements and of limited dimension. In fact, for a very large system to be effectively deployed, it must consider also extensive testing and simulations and a careful deployment, issues that as of now find no accommodation in Gaia.

Roadmap

Roadmap (Juan et al., 2002) focuses on building open systems giving special emphasis to the societal aspects of the multiagent system. To this end, it extend the original Gaia methodology (Wooldridge et al., 2000) by introducing use cases, a dynamic role hierarchy and additional models to describe the agent environment and the agent knowledge, as well as an interaction model based on AUML interaction diagrams.

As Gaia, roadmap is mainly focused on the analysis and design phases. The analysis phase contemplates the identification of: use cases, environment model, knowledge model, and of roles, protocols, and interactions models. Based on these models, the design phase aims at producing (as in Gaia) an agent model and a services model. As Gaia, roadmap lacks of support for detailed design, code and implementation, verification and deployment, thus designers have to adopt other methodologies to cover those aspects. However, the introduction of use cases in analysis can be considered as a sort of increased attention to requirements gathering, which is totally missing in Gaia.

We consider Roadmap as implicitly committing to a waterfall process model, due to the strict sequential nature of its analysis and design. Although roadmap documentation explicitly encourages an iterative approach to development, nothing is said in the methodology to support such iterations. However, as an improvement over Gaia, an explicit analysis of requirements, as provided by the adoption of a use case model, can make the sequential process of Roadmap more effective and reliable than that of Gaia in the presence of unstable or badly defined requirements.

Prometheus

Prometheus (Padgham and Winikoff, 2002) focus on building agents using BDI platforms, and on providing explicit and detailed guidelines and deliverables to industry practitioners with limited agent experience.

According to its proposers, Prometheus is a detailed and complete process for specify, designing and implementing multiagent systems. The methodology, per se, covers three phases: (i) system specification, (ii) architectural design, and (iii) detailed design.

The system specification phase covers the modeling of the system goals (that should result in one or more functionalities) as well as of a set of scenarios (modeling the system processing). The architectural design phase contemplates modeling agents, the system overview, and the inter-agent protocols (modeled using AUML). The detailed design phase focuses on developing the internal structure of each agent in term of its capabilities; each capability should be described by a set of descriptors (event, data, and plan). Prometheus also provides a hierarchical mechanism that allows designer to model multiple abstraction levels facilitating the design in great scale.

In addition, one should also consider that:

- The methodology is supported the Jack Development Environment (JDE) and the Prometheus Design Tool (PDT), two tools can support in the implementation and coding of the results of the Prometheus design phase;
- Guidelines for testing and debugging activities have been explicitly defined (Poutakidis et al., 2002).
- Activities related to requirement elicitation are partially covered by the analysis phase (i.e., requirements analysis is explicitly included, though as part of the analysis phase).

For which one can say that Prometheus cover, other than analysis and design, also requirements elicitation, coding, and testing.

The process defined by the Prometheus methodology is quite linear, proceeding sequentially from requirements elicitation to implementation and testing. A limited amount of feedback among phases may be identified only in the crosscheck relationship between the system overview (architectural design) and the agent overview (detailed design) models. For this reasons, Prometheus inherits the already identified limitations of waterfall-based methodologies.

In any case, the fact that Prometheus covers a significant number of phases of a software development process, it is more prone than, say, Gaia or Roadmap, of being easily adapted for exploitation in the context of other process models, e.g., evolutionary ones and the spiral one.

MaSE

The MaSE – Multiagent System Engineering (DeLoach et al., 2001) – methodology is organized in seven steps that do not have an immediate and explicit correspondence with the phases of the standard waterfall model: capturing goals, applying use cases,

refining goals into roles and their interactions, creating agent classes, constructing conversation, assembling agent classes, and designing the system.

According to our perspective, we consider that: goal capturing and the appliance of use cases partially cover the requirements elicitation phase; the analysis phase is based on the previous phase and includes the identification of roles, their tasks, and their interactions; then, the creation of the agent classes, of their conversation, and their assembling (including diagram for the deployment of the system), can be made corresponding to the design phase. Unlike the methodologies presented earlier, the MaSE methodology is limited to the development of closed agent systems, in which all agents are known a priori, and in which agents can trust each other during conversations. In fact, MaSE does not stress on the organizational aspects (i.e., the dynamic architectural issues) of the multiagent systems to be developed.

MaSE is supported by *agentTool* (Wood and DeLoach, 2000), a CASE tool supporting all MaSE's steps as well as code generation and automatic verification of inter-agent communications. Overall, the methodology covers all phases from requirements elicitation to implementation, and partially also the verification and testing activities.

According to the authors, MaSE has been conceived to be applied iteratively and incrementally. However, from all the documents available, the general process is presented like a sequential (waterfall) process, with no explicit description of iterated activities.

The fact that MaSE covers in a rather satisfactory way most of the phases of software development, can makes us think that MaSE could be indeed easily adapted – with all the advantages that this carries on – to be exploited in the context of evolutionary or spiral process models. Of course, this requires that suitable guidelines are defined to support the needed iterations. In any case, unless the methodology is properly extended to deal with the peculiar problems of open agent systems (which are the vast majority) its applicability would remain very limited.

AOR – Agent-Object-Relationship

AOR (Wagner, 2003) is a peculiar methodology, very different from the above presented ones, which contemplates the so called external models (i.e., models for the analysis) and the internal models (i.e., models for the design). The central abstractions of the AOR methodology are the so called "entities" (i.e., agents, events, actions, commitments, as well as ordinary objects) and special relationships to be defined among them that supplement the classical association, aggregation/composition and generalization/specialization relationships of UML models.

The external view (the analysis) aims at capturing the perspective of an external observer (i.e., of one or more so called "focus agent/s") to model the domain producing one or more of the agents, interaction frames, interaction sequences, interaction pattern diagrams. Then, for each focus agent, the internal view (the design) aims at capturing the functionalties of the system. The internal AOR model is then refined for each focus agent into an implementation model for the target language/platform. AOR modeling language (AORML) tools as well as a Microsoft Visio template are available for support AOR Modeling activities.

The overall process of AOR consists of 5 steps: *(i)* domain analysis (the external AOR model); *(ii)* transforming the external AOR model into an internal AOR model for the focus agent, and iterating this step for each focus agent that require an information system development; *(iii)* transforming the internal AOR models into database design models or logical data structure definition; *(iv)* refining the design models into

implementation models; *(v)* generating the target language code. Thus, one can consider AOR to cover requirements analysis, design, coding, and implementation phases.

The process model adopted is like an incremental waterfall with iterative work within some stages but no explicit iterations among phases. Thus, despite the very different nature, AOR shares the same advantages and drawbacks of MaSE.

3.2. Evolutionary and Incremental

In this category we can classify those methodologies that explicitly focus on an incremental and/or iterative approach among all the process. These include: OPM/MAS (Dori, 2002); MASSIVE (Lind, 2001); Ingenias (Gómez-Sanz and Pavón, 2002); Tropos (Bresciani et al., 2001; Giunchiglia et al., 2002); and PASSI with its extension Agile PASSI (Cossentino et al., 2004a; Cossentino et al., 2004b).

OPM/MAS

OPM/MAS - Object-Process Methodology for Multiagent Systems (Sturm et al., 2003) inherits its capabilities from object and process oriented paradigms. In particular, it is conceived as an extension to OPM (Dori et al., 2002). OPM considers that objects and processes – as the key main components of a software systems – are equally important to describe the system's function, structure, and behavior. It adopts a single graphical model to describe objects, processes, and their attributes (Object-Process Diagrams – OPD) and a corresponding automatically-generated English textual specification (Object-Process Language – OPL).

In OPM/MAS (Sturm et al., 2003) object and process are extended to include specific agent-oriented attributes, i.e.: for objects, organization, society, platform, rule, role, user, protocol, belief, desire, fact, goal, intention, and service; for processes agent, task, and messaging. Agent behaviors, in particular, are not necessarily encapsulated within objects, but may be modeled using stand-alone processes. OPM/MAS considers societal and organizational perspectives in multiagent system development, in which a society consists of rules, organizations and agent processes, while an organization consists of many roles and may exhibits many agent processes.

OPM/MAS supports the requirements, the analysis and the design phases of the development process, and a deployment diagram is included in the methodology. For the implementation and testing stages, CASE tools are under development.

With regard to the process model, OPM/MAS – as OPM – adopt a single notation to describe both the structure and the behavior of the components of a multiagent system. One of the distinguish characteristics is that this notation can be incrementally and selectively refined in deeper specification. It contemplates different scaling mechanisms, which are unfolding/folding, in-zooming/out-zooming and expression/suppression. These mechanisms overall facilitate to selectively and incrementally focus on a specific subset of issues, refining the description down to any desired level of detail, and thus helping in manage the complexity of a system model at any moment in the development process.

For these reasons, and despite the fact that the general process model is not made explicit in OPM/MAS, it possible to state that OPM/MAS is intrinsically suited for an evolutionary and incremental model, in which the various parts of a complex, very large-scale, multiagent systems, are incrementally refined as the development process progress. However, the potentials of OPM/MAS will be unfold in full only with the availability of proper tools for implementation and testing.

MASSIVE

MASSIVE - Multi Agent SystemS Iterative View Engineering (Lind, 2001) is a pragmatic method for the design and construction of multiagent systems.

MASSIVE is based on a view-oriented approach: during the development of a multiagent system, different phases can be executed focusing on different aspects of the systems. The considered views are: environment, task, role, interaction, society, architectural and system view. To exemplify: in the society view, the multiagent systems is considered as a structured collection of agents, organized according a particular organizational model; in the interaction view, the multiagent system is considered as an ensemble of interacting agents, in which various forms of competition and cooperation, as well as non-traditional forms of cooperation, may be identified.

The development methodology based on views offers various models of a multiagent system that can be used to incrementally define it. In particular, in MASSIVE, views are explicitly embedded in a process model inspired by stepwise refinement, the so called iterative view engineering. This is a product centered process model combining round-trip engineering (i.e., alternating software construction from specifications and reverse engineering to improve specifications) and iterative enhancement (i.e., promoting several cycles of the process to enhance a partitioned and incomplete software model).

Another characterizing part of MASSIVE is the so called "experience factory", which provides a conceptual framework for enabling a systematic learning process within an organization. This way it is possible to improve the development process and product models of a particular project according to the experience gained in the development process (so as to reach a higher maturity of the process).

Overall, the MASSIVE process covers analysis, design, implementation verification and testing, and deployment. No attention is paid to requirements elicitation.

In our opinion, MASSIVE is one of the few examples of a methodology for multiagent systems that explicitly takes care of the underlying process model in a very detailed way. The fact that it is based on a well-organized incremental process model and that it considers most of the relevant phases of multiagent systems development, makes MASSIVE a very promising approach, although its applicability may be possibly undermined by the lack of a proper requirements elicitation phase as well as by lack of adoption of notations (e.g., AUML) already well accepted in the multiagent systems community.

INGENIAS

INGENIAS (Gómez-Sanz and Pavón, 2002) build on previous work on the MESSAGE methodology (Caire et al. 2001a, Caire et al. 2001b). Its aim is to support multiagent systems development by generating multiagent systems specifications incrementally covering the analysis, design, and implementation phases of the development process. It adopts AUML as the basic notation to support the process.

INGENIAS explicitly applies the Unified Software Development Process – USDP (Jacobson et al., 1999). This is an iterative process model identifying two dimensions for the software developing process: time (phases of the life cycle), and content (models and other artifacts). Thus, the explicit process model it adopts is the so called stepwise refinement one, a specific form of an incremental process model.

INGENIAS identifies five meta-models to be exploited in development: agent model, interaction model, tasks and goals model, organization model, and environment model. Such meta-models (partially inherited from the MESSAGE methodology) allow the designers to incrementally define the architecture and the functionalities of the

multiagent system, by focusing on different points of view during development (in a similar way that MASSIVE does).

INGENIAS supports multiagent systems engineers using three elements: *(i)* a visual language for multiagent systems definition (GOPRR), *(ii)* integration with the phases and workflows of the USDP, and *(iii)* development tools: the meta-case METAEDIT+ (Lyytinen and Rossi, 1999).

Since INGENIAS adopts the USDP process model, it is iterative and incremental covering analysis, design, and coding and implementation phases. This suggests that INGENIAS, the same as MASSIVE, could be a very effective methodology for rapid development of agent system, with the additional advantage of providing standard notations and ready-to-use tools. Still, missing it the requirements elicitation phase, it can hardly be applied to critical systems design.

Tropos

The key characteristic of the Tropos methodology (Bresciani et al., 2001; Giunchiglia et al., 2002) is its strong focus on early requirements elicitation, the phase in which the domain stakeholders and their intentions are identified and analyzed. The requirements phase of Tropos is strongly influenced by the i* modeling framework (Yu, 1995). The main concepts on which Tropos is based are the "actors" with their goals, their plans, and the inter-dependencies.

All analysis of system requirements in Tropos is based on the goals that must be achieved by the system to be, and on the identification of the actors (whether humans or agents) that should be considered to achieved these goals. The process adopted by Tropos is basically one of analyzing goals on behalf of different actors, and is described in term of a non deterministic concurrent algorithm, including a completeness criterion. A few primary goals are analyzed from the perspective of its respective actor (typically humans at the beginning of the process), and as subgoals are identified, they are delegated to other actors (typically agents) or assumed by the actual actor. This analysis is carried out concurrently and normally implies iteration among different phases, especially between requirements elicitation and requirements analysis. Once all goals are identified and assigned to specific actors, the design phase aims at producing the organizational structure of the systems (i.e., identifying relationships between agents), and at detailing the specific characteristics of the composing agents. The implementation phase relies on appropriate AUML-based tools.

Overall, Tropos covers all the phases of the software development process up to the coding and implementation. The incremental iterative nature of the process, however, reduces to requirements elicitation, analysis, and design, ruling implementation out of the cycle. For this reason, we believe Tropos is very suitable to the development of those multiagent systems that rely on unstable or hard-to-be-identified requirements, and for which an incremental process must be followed before a stable design satisfying the requirements can be identified. However, it is definitely not suitable for those projects requiring rapid prototyping and quick delivery of products.

PASSI and Agile PASSI

PASSI (Cossentino et al., 2004a) is a methodology for multiagent systems development that covers all phases from requirements analysis to system implementation and deployment. The design activity is carried out adopting five sequential phases that explicitly take into account the need for incremental refinement.

In particular, PASSI covers the following phases of software development: *(i)* system requirements, to produce a use-case based description of the functionalities

required for the system and an initial decomposition of them accordingly to the agent paradigm; *(ii)* agent society, which is an analysis phase aimed at composing a model of the social interactions and dependencies among the agents of the solution. It includes the definition of the domain ontology and the specification of communications and agents' roles; *(iii)* agent implementation, which is a design phase aimed at modeling the solution architecture in terms of required agents, classes and methods. It is composed of both a structure definition and a behavior description of the whole system; *(iv)* code, the implementation phase aimed at modeling a solution at the code level. It is largely supported by patterns reuse and automatic code generation; *(v)* deployment, aimed at modeling the distribution of the system parts across a distributed platform. PASSI also includes a description of a possible testing approach divided in two different stages: the agent test, where each single agent is tested after its implementation, and the society test, where the multi-agent system is tested after deployment.

Although explicitly incremental, the great number of sequential phases in a cycle may require a long time before the first prototype code can be obtained in PASSI. Also, since the process is iterative both among the phases and in the whole software development process, this configures PASSI as a suitable in a application problems for which the coding phase can be positioned somehow late in the process, i.e., projects with a low level of changes in requirements.

In order to deal with smaller projects and more dynamic problems, the authors (Cossentino et al., 2004b) conceived an adapted version of PASSI, i.e., Agile PASSI which preserve the iterative and incremental nature but – by following the ideas of agile processes – lead to a quicker process more oriented to code delivery than to documentation production.

PASSI is an example of a complete methodology that not only pays the needed attention to the process model, but also recognizes that different application needs may require different types of processes, and accordingly face the problem of adapting the methodology to different agile process models.

3.3. Transformation

As an example of a methodology that can be somewhat considered committed to a transformation process model, we report here about DESIRE (Brazier et al., 2002), which also exhibit characteristics that could make it fall also in the evolutionary and incremental hat.

DESIRE

The compositional multi-agent design method DESIRE - DEsign and Specification of Interacting REasoning components (Brazier et al., 2002) supports the design of component based autonomous interactive agents considering conceptual design, instead of implementation level, and the system's specification exploiting knowledge-based techniques.

DESIRE views the individual agent and the overall system as compositional structures modeling both the inter-agent functionalities (requirements for the coordination and cooperation in the social interaction expressed in term of knowledge and reasoning skills) and the intra-agent ones (requirements for the tasks domain), as well as taking into account the processes and knowledge dimensions. DESIRE partially captures the organizational aspect of a multiagent system in term of task control rules (coordination) among agents at the knowledge level and in term of information links. However, it does not consider the organizational structure.

In DESIRE the following models are supported: *(i)* problem description; *(ii)* conceptual design; *(iii)* detailed design; *(iv)* operational design; and *(v)* design rationale. Also, to improve the reusability, the methodology offers designers a set of generic models (i.e. generic co-operative agent model, generic model of a BDI-agent, generic model for diagnostic task, for design task, generic model of reasoning path, etc.). It is worth to note that there is no fixed sequence of design.

DESIRE covers the phases form requirements elicitation up to the verification and testing (partially). In fact, the high level modeling environment of DESIRE allows engineers to automatically generate prototypes of multiagent application from the detailed design. On the basis of these partial prototypes, new designs and prototypes are iteratively generated and examined. In this sense DESIRE can be considered to adopt an evolutionary model.

However, during the problem specification phase, informal requirement are incrementally transformed into formal ones, to facilitate code generation and the verification process. Due to the formalization of the requirements, the verification process is done by a mathematical proof and doing so, the verification and testing phase is partially covered. For this reasons, we can also consider DESIRE to adopt, at least for some of its phase, a transformation model.

We have already stated in Section 2 what we consider the limitations of the transformation model for practical industry applicability. Still, the possibility enforced by DESIRE of somewhat integrating in an incremental model the additional power of a transformation model represents a potentially promising approach.

3.4. Spiral

We found a single methodology that could be clearly included in this class, i.e., the MASCommonKADS one (Iglesias et al., 1997).

MAS-CommonKADS

MAS-CommonKADS (Iglesias et al., 1997) extends the models defined in *CommonKADS*, adding techniques from object-oriented methodologies (like Object Modeling Technique – OMT, Object Oriented Software Engineering – OOSE, and Responsibility Driven Design - RDD) and from protocol engineering to describe the agent protocols.

The methodology starts with a conceptualization phase which is an informal phase used to collect the users' requirements and to obtain a first description of the system from the user's point of view. Consequently, MAS-CommonKADS partially covers the requirement elicitation phase by means of the use case model. For the following analysis and design phase, MAS-CommonKADS defines the following models: *(i)* agent model; *(ii)* task model; *(iii)* expertise model; *(iv)* coordination model; *(v)* organization model; *(vi)* communication model; and *(vii)* design model, which contemplates the application design, architecture design and platform design.

According to its proposers, MAS-CommonKADS also considering the coding unit testing, integration and global testing, and implementation and maintenance stages, however, we have not found supporting documentation for those phases.

The process model adopted for small projects is a waterfall-like based on the reuse of components preciously defined, while for large projects MAS-CommonKADS adopts the same process model of the CommonKADS methodology, that is, the spiral model.

Although rather obsolete, and possibly hardly applicable to modern multiagent systems scenario, the MAS-CommonKADS methodology has the advantage of showing that a spiral process model can be applied in the context of multiagent systems. In

addition, it points out (as PASSI does) that the same methodology can be effectively adapted and applied in the context of different process model (i.e., a waterfall for small projects, and a spiral for complex and risky projects).

3.5. Summary

Most of the AOSE methodologies analyzed (those presented in this section as well as those that we have excluded from the presentation for the sake of length limitations) adopts either a waterfall-like or an evolutionary/incremental model.

In particular, from the above analysis we can state that:

- Those methodologies that do not make any explicit reference to the process model, end up in promoting at rather standard waterfall process model or – more rarely – a rough incremental process model;
- Those methodologies paying more care to the process model issue end up in explicitly proposing an incremental process model.

Summarizing, we can state that the need for incremental process models is widely recognized in the community.

A very few methodologies adopts a transformation-like model (here we have mentioned DESIRE). Although other attempts in transforming informal specification into code by mean a transformation process have been explored so far (consider e.g., the work of d'Inverno and Luck (d'Inverno and Luck, 1997) using Z schemas), those efforts are to be considered single methods and notations more than complete methodologies. Nevertheless, it is our opinion that the use of formal model for multiagent systems development will notably increase in the future (as already stated in Section 2).

Spiral models too have encountered a very limited success. Very likely, the reason is that a few complex industrial projects (involving high risks) have been so far carried out. Thus, the need to anticipate and possibly eliminate the risks associated with complex software development projects in agent based development have simply not emerged. Still, we expect the spiral model to increase its role in the future, with the increase of multiagent systems to be developed in real-world complex setting.

4. OPEN ISSUES

As already discussed, up to date, researchers and practitioners in AOSE have paid more attention to methodologies rather than the whole software development process. Methodologies play indeed a very important role, but require to be put into the context of a process model. With regard to the latter, in addition to the specific issues discussed in the previous section, several further issues strictly legated to process models may represent interesting challenges for researchers and practitioners in this community.

Hereinafter, we present some of them considering four complementary assessment directions: on the process models, focusing on the need of specific agent-oriented agile process models (Sub-section 4.1); on the methodologies, focusing on the need of multi-perspective approaches (Sub-section 4.2); on the methods and the meta-models for agent-oriented development (Sub-section 4.3); on the tools supporting the process of agent-based development (Sub-section 4.4); on the evaluation and improvement of the process and the resulting product quality (Sub-section 4.5).

4.1. Extreme Programming of Multiagent Systems

Section 3 has already outlined the need for incremental, and possibly very agile processes for software systems and multiagent systems development. However, some further considerations on this are to be reported.

It is a matter of a fact that most industrial practice of software development – due to the lack of time, strict schedules for delivery, no time and resources to spend in documentation activity – end up in being unstructured, frenetic, and missing at all some kind of organization in human resources. The most innovative trend in agile approaches to software development, namely Extreme Programming (XP) is an interesting attempt to start from the above state, and proposes a very agile structure that can bring some engineering flavor to the process without forcing to spend too many resources, preserving the need for quick delivery and, at the same time, ensure more reliability. The industrial world has positively reacted to this proposal and the new approach is rapidly gaining a relevant industrial acceptance for the development of conventional types of small- medium-size software in projects affected by uncertain or changing requirements (Succi and Marchesi, 2001).

To the best of our knowledge, the only proposal towards and XP approach in agent-based development is that of Knublauch (2002), proposed with an explicit support for change and rapid feedback, and assumed to be useful for small to medium-sized projects. The approach consists in building and maintaining in a cyclic fashion two main models: *(i)* a process model (for the design of agent scenarios); and *(ii)* the executable agent source code including automated test cases. The process model aims at capturing and clarifying requirements, visually documenting agent functionalities, and facilitating the communication with end users. Using specific tools it is possible to automatically generate source code, thus introducing the next phases (coding and testing), and focused on interactions among agents and agent life-cycle management. Finally, the cyclic development process consists in switching between the implementation and process models updating them arbitrarily.

Although it is a pioneering proposal potentially very interesting, it has several limitations – which can be considered as current general limitations of the agent technology rather than limitations of the proposal itself. First, the complexity of potential agent interaction scenarios and the emerging behaviors within a multiagent may make pre-planning very hard (Lind, 2001), which is in sharp contrast with the very foundations of the XP philosophy, stating that implementation and evaluation of executable code must have priority over a comprehensive documentation. Second, being XP strongly concerned with rapid prototyping and testing, the lack of appropriate tools for the testing of multiagent systems (making it impossible to systematically evaluate specific agent-oriented problems such as the respect of social rules, the correct enactment of collaboration strategies etc.) represents a serious issue.

We strongly believe that agile and XP-like processes for agent-based development are needed to improve acceptance of agent-technology by industry. Still, further researches in the AOSE area are needed to pave the way to XP-oriented agent-based development.

4.2. Multi-perspective Approaches

Another potentially interesting direction to improve the effectiveness and reliability of agent-based development is to focus on different perspectives during the development process. With the term perspective, we mean an abstract representation of the system, enabling the evaluation of one or more features of the same system, thus highlighting some aspects of current interest, while hiding all the others that are not

interesting from that specific point of view. This is an application of different fundamental principles of the software engineering (i.e. separation of concerns, horizontal modularity, etc.) may help a more exhaustive comprehension of the system to be in all of its different aspects.

In mainstream software engineering, the application of this approach has oriented the development process to the recognition of the importance of some paradigms that were associated to the software design and construction activities. For instance, structured analysis and design approaches specify the system to be adopting two main perspectives: the data flow and the data structure. On the other side, object orientation defines the system considering the objects (and their description) perspective, the (instantiation) relationships among objects and classes, and, finally, the interactions among different classes (or objects). Also in the artificial intelligence area, in the construction of expert system, two mainly perspectives are considered: the knowledge representation and the inference skills.

Recently, in the AOSE arena, this multi-perspective idea has been adopted – to different extent – in a number of proposals (see for example Massive and INGENIAS, to refer to two methodologies presented in this paper) and has also been advocated in a different paper by the authors (Cossentino and Zambonelli, 2004).

While it is easy to recognize the importance of multiple perspectives in order to achieve a thorough comprehension of the system (especially when dealing with multiagent systems that are often used to implement complex and distributed solutions), it is not easy to obtain a multi-perspective design process that conjiugates both quality and cost parameters. Quality pursuing would bring to increment the number of different perspectives but the need of avoiding the introduction of colliding specifications in different views pushes to adopt a specific design tool support for maintaining their coherence. Both of these aspects (an increment in the number of perspectives and the coherence check performed by a specific design tool) originate an increment in the project costs (and time) and therefore limit the possibility of diffusely applying this approach.

Accordingly, we believe further investigations are needed to make the multi-perspective approach practical, i.e., as a way to reduce complexity rather than increase the costs of software development.

4.3. Meta-Models, Meta- Methodologies, and Method Engineering

Actually, several works (Cossentino and Seidita, 2004c; Odell et al., 2003; Henderson-Sellers, 2003) are focusing on the identification of appropriate meta-models for AOSE methodologies and process models, where a meta-model is intended as rational analysis and identification of the abstractions used in multiagent system development. Those efforts aim at may be very interesting to clarifying and possibly try to unifying the different abstractions adopted in existing by the methodologies and the process models and also at identifying which relationships may exist among them. This may be used to better understand the real usefulness of the abstraction and to improve or unify processes and methodologies. Also, it may help researchers and practitioners to identify and develop conceptual instruments and practical tools that could enable tools for an efficient the processes management. However, since it is quite hard to synthesize the better of all existing proposals (i.e., of all methodologies and processes), in this approach still subsists the risk of producing a result that is too complex and not enough affordable.

In this direction, we can cite the contributions coming from the FIPA Methodology Technical Committee (Cossentino et al., 2003) and the OPEN framework (Henderson-Sellers, 2003) that adopt the method engineering paradigm (Saeki, 1994), more appropriately called situational method engineering (Ter Hofstede and Verhoef, 1997). In this approach the development process is composed by assembling pieces (method fragments) of it from a repository of methods built up taking pieces from existing processes/methodologies. Each method fragment is mainly composed by three elements: *(i)* the process to achieve fragment objectives; *(ii)* the artifacts to be produced; and *(iii)* the roles played by the involved people.

The process composition by reusing existing parts may be seen as an application of the compositionality software engineering principle, according to the roman idea of "divide et impera", strongly related with modularity, incrementality, abstraction, and separation of concerns, that are principles frequently advocated by software engineering authors (see for example (Ghezzi et al. 1991)). Also, the relationships among components may be associated to the object oriented aggregation relationship.

The complete method engineering process could start from the selection of the elements that compose the meta-model of the multiagent system. Then the development process is composed by selecting proper fragments from the repository. In FIPA those activities are supported by a Computer Aided Method Engineering – CAME tool. Based on the constructed process, the development team could perform the established activities thus obtaining a model of the multiagent system.

To harmonically integrate all the different fragment of methods, engineers will need guidelines; up to now, some guidelines are available, for example in the OPEN framework, about the adequate use of a particular method in a specific context, that is, about the correct use of a single method fragment (component). Some kind of patterns, i.e. design or architectural patterns, have been proposed in specialized literature, but to really accomplish with the method engineering purpose, engineers need more "methodological patterns" for the thorough process resulting by the method's integration.

As we have seen, the idea is very simple and very good. However, engineers have a big challenge in applying it. It could seem to be easy to choose from a repository the more adequate fragments of models and methods but instantiating a new process by composition is currently one of the more difficult and time-consuming work of the method engineering approach. In effect, the method engineer has to clearly perceive the context (including the organization) and the target software problems in order to select the appropriate method fragments to use in the specific project.

Complementarily on the need of methodological guidelines or patterns, another relevant question grows up. Is it easier to put together a set of different methods and techniques from substantially different methodologies or to adapt an existing methodology or process model that may represent a more consolidate approach?

As almost in every discipline, this decision as well as the harmonic integration of methods fragments or the adaptation of an existing methodology or process model is very hard if engineers have not expertise. Thus, a correlated issue consists in the way that could be used to accumulate and share the expertise of the (few) already skilled method engineers.

Other interesting attempts are trying to propose new processes as a synthesis of the best models proposed by previous ones. The idea is to find out the minimum common ground between different processes and to extend it to a general approach by adopting the best features of each considered process in a consistent way. Some example of this trend are Skeleton (based on Roadmap and Prometheus) (Juan et al., 2003), INGENIAS

(Gómez-Sanz and Fuentes, 2002), MESMA (Cuesta et al., 2002), etc. However, this practice still not guarantees that the results would be more useful than the previous process they extended. In effect, it is practically impossible to take into account all the best contributions of the already existing methodologies and the frequently newly proposed ones. Moreover, probably it is impossible to define the "panacea" process that better covers all the multiple possible project cases and multiagent applications. Thus, every syncretism is destined to be partial and, from the point of view of the process model, to suffer the some drawback of the original approach it extends.

4.4. Tools

When developing an agent-based system, several tools are necessary during the different stages of the process (here we will classify them in three categories: design, implementation and deployment); in the design phase we need tools allowing the specification of the system (also using formal languages when needed), its validation and (possibly) offering good automatic code generation capabilities; for implementing agents we need agent-oriented languages or (as it often happens) some object-oriented library that allows the code-level implementation of the agent abstractions; this phase should also be supported by rapid development environments also offering multiagent systems testing features; finally agent platforms are necessary for the system deployment and the realization of efficient communication channels among the distributed portions of the agent society. Because of the scope of this paper, in the following we will only discuss design tools and their related open research issues.

Obviously the different phases of the software design process can be covered using separate tools (sometimes with some level of inter-operability) or a unique environment. The problem of the first choice is that using different tools the designer needs to make them communicate and this is often in contrast with their data exchanging capabilities; this could bring to time consuming manual operations that could likely introduce errors in the design. On the contrary, using the other option (a unique tool for the whole design process), the designer is supposed to use a unique well integrated tool, usually quicker to learn than several different ones and without any problem in forwarding design information from one stage to another. The inconvenience of such tools is that they are often rigid and force the user to adopt their way of thinking thus limiting the possibilities of applying new approaches; this often limits with the possibility of successfully applying an advanced agent-oriented design process.

Application fields for design tools spread from requirements elicitation to design, testing, validation, version control, configuration management and reverse engineering.

Dealing specifically with design tools, we can classify them in three different categories: CASE, CAME, CAPE tools. The CASE acronym means Computer Aided Software Engineering and it names tools that could be used to design with the most different approaches. Usually they well support the modeling activities and only constrain the designer in the choice of the system modeling language (for instance UML), and, when code generation is possible, on the list of supported coding languages. The main limit of these tools is that they are not aware of the adopted method (in terms of work to be done) but they are only concerned with the representation of some (often not coordinated) views of the system. While these tools are very diffused in the object-oriented systems development context, at our knowledge, there exists no agent-oriented tool that has reached an industrial quality level that could be compared to corresponding object-oriented software; several examples of research-level CASE tools are available for the design of multiagent systems but they usually support only one specific

methodology. The importance of having tools conceived for supporting multiagent systems development, descends from the specific needs in modeling a multiagent system and the different (but coordinated) levels of abstractions that are necessary.

As regards CAME tools (CAME stands for Computer Aided Method Engineering), they are conceived to support methods rather then design. This means that they could be programmed (or configured) to be aware of the semantics and relationships of the concepts involved in the development of a specific category of systems and specific views can be prepared to represent the model. They do not adopt any specific software development process model (they are not even aware of its existence because they are only concerned with the drawing of the different aspects of the model separately) and therefore the designer could freely work on the different views even violating the prescribed process without any warning from the tool. They are the most natural tools for implementing a methodology (seen as a collection of methods). No specific CAME tool for agents is reported in literature; although the flexibility of such a kind of instruments (often classified as meta-tools) allows a high level of adaptability to the most different needs, the existing tools (object-oriented in their conception) inherently lack of an agent-orientation (for instance they do not intrinsically support concepts like ontology, knowledge, and social rule) and this causes a not natural use of them in the multiagent systems design.

One of the intrinsic limits of CAME tools (the lack of process awareness) is overcome by CAPE (Computer Aided Process Engineering) tools that are aware of the adopted process model (or could even be used to design it) and coordinate the different stages of the design in order to respect its prescriptions. This category of tools is, at our knowledge, totally unexplored in the field of agent-oriented software development and their growth is as desirable as the directly related development of specific agent-oriented processes.

4.5. Evaluation and improvement of processes and resulting products quality

A possible way to let accessible the expertise is to insist on evaluation frameworks that could highlight the advantages and drawback of agent-based methodologies and process models in particular contexts. In this sense, different authors have proposed interesting works on the evaluation of methodologies (Shehory and Sturm, 2003; Cernuzzi and Rossi, 2002; Hoa Dam and Winikoff, 2003; Cuesta et al., 2003; etc.). However, at our knowledge no work has focused on the evaluation of process models and we think that more effort is still needed in evaluating methodologies, specific methods and techniques in AOSE. In effect, the great majority of the proposed works are centered on qualitative evaluation, while engineers need of more quantitative results that may facilitate comparative analysis and the selection of specific methods.

5. CONCLUDING REMARKS

Since the very beginning of software engineering researches, a variety of software process models have been proposed, from sequential waterfall-like to evolutionary and transformation-based ones, with the goal of identifying effective, reliable, and reproducible ways to produce software. In the community of software engineering, there is now a general consensus that for most real-world industrial projects the pervasive waterfall model should be better replaced by more flexible and iterative approaches, such as evolutionary and spiral ones. Also, it is an acknowledged fact that no single general-purpose process model can be effective for all projects, and that different

commercial and engineering needs may be satisfied by different process models. In addition, software processes cannot be defined and established once and for ever, they need to be continuously assessed and improved.

The above considerations imply that a major duty of a software engineer – other than designing software by applying methodologies – is to apply its expertise to identify the most appropriate process model for any specific situation, and put this model at work. Unfortunately, despite the above understanding, this paper has outlined that current researches in the area of AOSE and of AOSE methodologies underestimate the importance of the process model in multiagent system development. In most of the cases, an AOSE methodology gets proposed without any explicit reference to the underlying process model.

It is our hope that the analysis and the discussions reported in this paper may somewhat clarify about the importance of process models in agent-oriented software development, and may be of inspiration for process-oriented researches in the AOSE community. In addition to the open issues identified in this paper, further issues related to the engineering of very large collectives of distributed agents exhibiting complex and emergent behaviors, and to the analysis of the innovative process models that could suit these kinds of systems, would be worth to be investigated (Zambonelli and Omicini, 2004).

REFERENCES

Bauer, B., Müller, J., and Odell, J., 2000. Agent UML: A Formalism for Specifying Multiagent Software Systems. In: Ciancarini, P., and Wooldridge, M. (Eds.) Agent-Oriented Software Engineering - Proceedings of the First International Whorkshop (AOSE-2000). Springer-Verlag, Berlin (Germany) , pp. 91-103

Beck, K., 1999. Extreme Programming Explained: Embrace Change. Addison-Wesley, Boston, MA (USA)

Boehm, B., 1998. A Spiral Model of Software Development and Enhancement. *IEEE Computer*, Vol. 21, Nº 5, May, 1988, pp. 61-72

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J., 2001. A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. In: Proceedings of the 5th International Conference on Autonomous Agent*s*. ACM Press, Montreal (Canada), pp. 648-655

Brazier, F., Jonker, C., and Treur, J.**,** 2002. Principles of Component-Based Design of Intelligent Agents. *Data and Knowledge Engineering*, vol. 41, No. 2, pp. 1-28

Caire, G., Chainho, P., Evans, R., Garijo, F., Gómez Sanz, J., Kearney, P., Leal, F., Massonet, P., Pavón, J., 2001a. Agent Oriented Analysis Using MESSAGE/UML. Proceedings of Agent-Oriented Software Engineering – AOSE 01, May 2001, Montreal (Canada), pp. 101-107

Caire, G., Leal, F., Chainho, P., Evans, R., Jorge, F.G., Pavón, J., Kearney, P., Stark, J., and Massonet, P., 2001b. Project p907, deliverable 3: Methodology for Agent-Oriented Software Engineering. Technical Information Final version, European Institute for Research and Strategic Studies in Telecommunications (EURESCOM), September 2001. Available from http://www.eurescom.de/ public-webspace/P900-series/P907/D3_nalReviewed.zip.

Cernuzzi, L. and Rossi, G., 2002. On The Evaluation Of Agent Oriented Methodologies. Proceedings of the OOPSLA 02 - Workshop on Agent-Oriented Methodologies, November 2002, Seattle (USA), pp. 21-30

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), Sept. 2004, Beijing (China),

Ciancarini, P. And Wooldridge, M., 2001. Agent-Oriented Software Engineering. Proceedings of the 1st International Workshop on Agent-Oriented Software Engineering, Springer Verlag, LNCS, Vol. 1957, pp. 1-24

Cossentino, M., Hopmans, G., Odell, J., 2003. FIPA Standardization Activities in the Software Engineering Area. Workshop on Object and Agents (WOA'03) Sept, 10-11, Cagliari (Italy),

Cossentino, M. and Sabatucci, L., 2004. Agent System Implementation in Agent-Based Manufacturing and Control Systems: New Agile Manufacturing Solutions for Achieving Peak Performance. Paolucci M. and Sacile R. editors. CRC Press, April 2004

Cossentino, M. and Seidita, V., 2004. Composition of a New Process to Meet Agile Needs Using Method Engineering. *Software Engineering for Large Multi-Agent Systems* vol. III. LNCS Series, Elsevier Ed..

Cossentino, M. and Zambonelli, F., 2004. Multiagent Systems Development from the Autonomy Perspective, *Computational Autonomy*. LNCS Series No. 2969, Elsevier Ed.

Cuesta, P., Gómez, A., González, J.C., and Rodríguez, F., 2002. The MESMA Approach for AOSE. Proceedings of Fourth Iberoamerican Workshop on Multi-Agent Systems (Iberagents'2002), at IBERAMIA'2002, the VIII Iberoamerican Conference on Artificial Intelligence, November 11-12, 2002, Malaga (Spain),

Cuesta, P., Gómez, A., González, J.C., and Rodríguez, F., 2003. A Framework for Evaluation of Agent Oriented Methodologies, Proceedings Taller de Agentes Inteligentes en el Tercer Milenio (CAEPIA'2003), November 2003, San Sebastián (Spain)

DeLoach, S., Wood, M. and Sparkman, C., 2001. Multiagent Systems Engineering. *International Journal of Software Engineering and Knowledge Engineering*, vol. 11, No. 3, pp. 231-258

Dori, D., 2002. Object-Process Methodology – A Holistic System Paradigm. Springer, Berlin, Heidelberg, New York

d'Inverno, M., and Luck, M., 1997. Development and Application of a Formal Agent Framework. Proceedings of the First IEEE International Conference on Formal Engineering Methods, November 12-14, 2002, Hiroshima (Japan), pp. 222-231

Fuggetta, A., 2000. Software Process: a Roadmap. Proceedings of the Conference on the Future of Software Engineering, June 4-11, 2000, Limerick (Ireland), ACM Press, New York (USA), pp. 25-34

Gamma, E. and Beck, K., 2000. JUnit: A regression Testing Framework. http://www.junit.org

Ghezzi, C., Jazayeri, M., and Mandrioli, D., 1991. Fundamentals of Software Engineering. Prentice Hall International, Upper Saddle River, NJ (USA)

Gilb, T., 1988. Principles of Software Engineering Management, Addison-Wesley, Boston, MA (USA)

Giunchiglia, F., Mylopoulos, J. and Perini A., 2002. The Tropos Software Development Methodology: Processes, Models and Diagrams. Proceedings of Agent-Oriented Software Engineering (AOSE-2002), July 2002, Bologna (Italy), pp 63-74

Glaser, N. and Haton, M., 1996. Experiencing in Modelling Statistical Process Control Knowledge. Proceedings of the 12th European Conference on Artificial Intelligence – ECAI 1996, August 11-16, 1996, Budapest (Hungry), pp. 313-317

Gómez-Sanz, J. and Pavón, J., 2003. Agent Oriented Software Engineering with INGENIAS. Proceedings of the 3rd Central and Eastern Europe Conference on Multiagent Systems, Springer Verlag, LNCS 2691, pp. 394-403

Hoa Dam, K., and Winikoff, M., 2003. Comparing Agent-Oriented Methodologies. Proceedings of Agent Oriented Information Systems-AOIS'03, July 2003, Melbourne (Australia), pp. 78 - 93

Iglesias, C., Garijo, M., González, J.C. and Velazco, J.R., 1997. Analysis and Design of Multiagent Systems using MAS-CommonKADS. In: Singh, M., Rao, A.S. and Wooldridge, M. (Eds.), Intelligent Agent IV, Springer, LNCS 1365, pp. 312-328

Iglesias, C., Garijo, M. and González, J.C., 1999. A survey of Agent-Oriented Methodologies. In: Muller, J.P., Singh, M., and Roa, A.S. (Eds.), Intelligent Agent V, Proceeding of ATAL-98, Springer, LNCS 1555, pp. 317-330

Kelly, S., Lyytinen, K. and Rossi, M., 1996. MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. in Advanced Information Systems Engineering, Proceedings of the 8th International Conference CAISE'96 (eds. P. Constapoulos, J. Mylopoulos, Y. Vassiliou), Springer-Verlag, pp. 1-21

Knublauch, H., 2002. Extreme Programming of Multi-Agent Systems. Proceedings of the First International Conference on Autonomous Agents and Multi-Agent Systems - AAMAS '02, ACM Press, July 15-19, 2002, Bologna (Italy), pp. 704-711

Kruchten, P., 1998. The Rational Unified Process: An Introduction. Addison-Wesley , Boston, MA (USA)

Jacobson, I., Rumbaugh, J. and Booch, G., 1999. The Unified Software development Process. Addison-Wesley, Reading, MA (USA)

Jennings, N. R., 2001, An Agent-Based Approach for Building Complex Software System, Communications of the ACM, Vol. 44, No. 4, pp. 35-41.

Juan, T., Pearce, A. and Sterling, L., 2002. ROADMAP: Extending the Gaia Methodology for Complex Open Systems. Proceeding of the First International Conference on Autonomous Agents and Multi-Agent Systems - AAMAS '02, July 15-19, 2002, Bologna (Italy), pp. 3-10

Juan, T., Sterling, L. and Winikoff, M., 2002. Assembling Agent Oriented Software Engineering Methodologies from Features. Proceedings of the First International Conference on Autonomous Agents and Multi-Agent Systems-AAMAS'02, Third International Workshop on Agent-Oriented Software Engineering AOSE-2002, July 15, 2002, Bologna (Italy), pp. 161-172

Lahlouhi, A. and Sahnoun, Z., 2002. Multi-Agent Methodologies' Incoherencies. Proceedings of the OOPSLA 2002, Workshop on Agent-Oriented Methodologies, November, 2002, Seattle (USA), pp. 64-73

Lahlouhi, A., Sahnoun, Z., Refrafi, A., Azizi, S., Khelifa, H.Z., Mimi, F., Kahloul, L., and Attaoua, Z., 2002. MASA-Method: A Multi-Agent Development Methodology". 6th World Multi-Conference on Systemic, Cybernetics and Informatics – SCI 2002, Software Engineering of Multi-Agents Systems – SEMAS 2002, July, 2002, Orlando (USA)

Letelier, P., Sánchez, P., Ramos, I., and Pastor, O., 1998. *OASIS version 3.0: Un Enfoque Formal para el Modelado Conceptual Orientado a Objetos*, 84-7721-663-0, UPV Publication Service, SP-UPV, 98-4011

Lind, J., 2001. Iterative Software Engineering for Multiagent Systems, the MASSIVE Method. Springer Verlag, New York, Secaucus, NJ, USA

Padgham, L. and Winikoff, M., 2002. Prometheus: A Methodology for Developing Intelligent Agents. Proceedings of the First International Conference on Autonomous Agents and Multi-Agent Systems - AAMAS '02, Third International Workshop on Agent-Oriented Software Engineering AOSE-2002, July 15, 2002, Bologna (Italy), pp. 135-146

Pastor, O., Insfran, E., Pelechano, V., Romero, J., and Merseguer, J., 1997. OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods. Conference on Advanced Information Systems Engineering (CAiSE), Is International, 9th, 1997, Barcelona (Spain), Springer, LNSC, 3-540-63107-0, 1250, pp. 145-159

Paulk, M., Weber, C.V., and Curtis, B., 1995. The Capability Maturity Model for Software. Addison Wesley, Reading (MA).

Perry, D., Staudenmayer, N., and Votta, L.G., 1994. People, Organizations, and Process Improvement. *IEEE Transaction on Software Engineering*, vol. 11 , No. 4, pp. 36 - 45

Poutakidis, D., Padgham, L. and Winikoff, M., 2002. Debugging Multi-Agent Systems Using Design Artifacts: the Case of Interaction Protocols. Proceedings of the First International Conference on Autonomous Agents and Multi-Agent Systems - AAMAS '02, July 15-19, 2002, Bologna (Italy), pp. 960-967

Saeki, M., 1994. Specifying Software Specification and Design Methods. International Conference on Advanced Information Systems Engineering, Utrecht (NL), LNCS No. 811.

Shehory, O., and Sturm, A., 2001. Evaluation of Modeling Techniques for Agent-Based Systems. Proceedings of the Fourth Intrenational Conference on Autonomous Agents, Montreal (Canada), pp. 624-631

Sturm, A., Shehory, O., 2003. A Framework for Evaluating Agent-Oriented Methodologies. Proceedings of Workshop on Agent-Oriented Information Systems – AOIS 2003, 5th International Bi-Conference, July 14, Melbourne, (Australia), October 13, 2003 and Chicago, IL (USA), 2003, LNCS 3030, pp. 94-109.

Sturm, A., Dori, D., Shehory, O., 2003. Single-Model Method for Specifying Multi-Agent Systems. Proceedings of the Second International Conference on Autonomous Agents and Multi-Agent Systems - AAMAS'03, July, 2003, Melbourne (Australia), pp. 121-128

Succi, G. and Marchesi, M., 2001. Extreme Programming Examined. Addison-Wesley, Reading (MA).

Ter Hofsted, A.H.M., and Verhoef, T.M., 1997. On the Feasibility of Situational Method Engineering. *Information Systems*, vol. 22, No. 6/7, pp. 401-422

Wagner, G., 2003. The Agent-Object-Relationship Metamodel: Towards a Unified View of State and Behavior. *Information Systems*, Vol. 28, No. 5, July, 2003, Elsevier, pp. 475-504

Wood, M., and DeLoach, S.A., 2000. An Overview of the Multiagent Systems Engineering Methodology. Ciancarini, P., and Wooldridge, M. (Eds.) Agent-Oriented Software Engineering - Proceedings of the First International Workshop (AOSE-2000). Springer-Verlag, Berlin (Germany), pp. 207–221

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*, Vol. 3, No. 3, pp. 285-312

Yu, E., 1995. Modelling Strategic Relationships for Process Reengineering. PhD thesis, University of Toronto, Department of Computer Science

Zambonelli, F., Wooldridge, M. and Jennings, N. R., 2003. Developing Multiagent Systems: The Gaia Methodology. *ACM Transaction on Software Engineering and Methodology*, vol. 12, No. 3, pp. 417-470

Zambonelli, F. and Omicini, A., 2004. Challenges and Research Directions in Agent-Oriented Software Engineering. *Journal of Autonomous Agents and Multiagent Systems*, vol. 9, No. 3, Kluwer Academic Publishers, pp 253-283