SPEM on test: the SODA case study

Elena Nardini
Alma Mater Studiorum – Università di Bologna
Cesena, Italy
elena.nardini@unibo.it

Ambra Molesini
Alma Mater Studiorum – Università di Bologna
Bologna, Italy
ambra.molesini@unibo.it

Andrea Omicini
Alma Mater Studiorum – Università di Bologna
Cesena, Italy
andrea.omicini@unibo.it

Enrico Denti
Alma Mater Studiorum – Università di Bologna
Bologna, Italy
enrico.denti@unibo.it

ABSTRACT

In the Software Engineering (SE) research field, several efforts are underway aimed at developing appropriate *meta-models* for SE methodologies. Meta-models are meant to check and verify both the software development process and the completeness and expressiveness of methodologies. In this context, in order to provide a uniform way to represent, compare and reuse methodologies, Software Process Engineering Meta-model (SPEM) – an OMG object-oriented standard – is a natural candidate.

In order to put the SPEM meta-modelling power to test, and emphasise its benefits and limitations, in this paper we apply SPEM to a more articulated context than the object-oriented one where it was initially conceived – that is, Agent-Oriented Software Engineering (AOSE) methodologies. In particular, we take the SODA methodology as a significant case study in order to assess strengths and limitations of SPEM, given the peculiar SODA focus on the modelling and engineering of (i) social issues and (ii) application environment – essential aspects in the engineering of complex software systems.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design—*Methodologies*; D.2.18 [**Software Engineering**]: Software Engineering Process—*Process definition, Software process models*

General Terms

Design

SAC'08 March 16-20, 2008, Fortaleza, Ceará, Brazil

Keywords

Multiagent System, Agent-Oriented Software Engineering, AOSE Methodology, SODA, SPEM

1. INTRODUCTION

The "silver bullet", one-size-fits-all methodology is nowadays a recognised *chimera* in the Software Engineering field. Quite naturally, each SE methodology has its own approach and peculiarities, its pros and cons—and none is general enough to fit with any possible application scenario. As a consequence, it has been observed [7] that software designers tend to define/refine their own problem-specific methodology, as an *ad hoc* tool specially suited to the typical application scenarios of interest. Of course, such methodological approaches can be hardly reused in different contexts, or for different problems, without significant changes: in the overall, this results in considerable costs of the system development process.

For these reasons, research efforts are ongoing meant to define a unified meta-model, aimed at representing the existing methodologies in a uniform way, so as to promote their mutual comparison, their composition and reuse—this area is sometimes referred to as *Method Engineering* [5, 25]. SPEM (Software Process Engineering Meta-model, [26]) and OPF (Object-oriented Process,Environment and Notation Process Framework, [24]) are two key references for this purpose: as it could be expected, both were conceived for an object-oriented context, since most current methodologies adopt this paradigm as the reference one. In particular, SPEM seems a natural candidate for representing the meta-models of Software Engineering methodologies, both because it is an OMG standard, and because it is based on formal descriptions that can lead to consistent, comparable models: so, an interesting challenge is to test its applicability to other, non object-oriented Software Engineering domains.

In this paper, in particular, we explore its applicability to the Agent-Oriented Software Engineering (AOSE) domain, whose abstractions and mechanisms are particularly suited to the design and development of complex software systems. While some AOSE methodologies have already been modelled in SPEM by the FIPA Methodology Technical Committee [8, 11], here we mean to exploit SPEM to model the SODA methodology process, taken as a significant case study

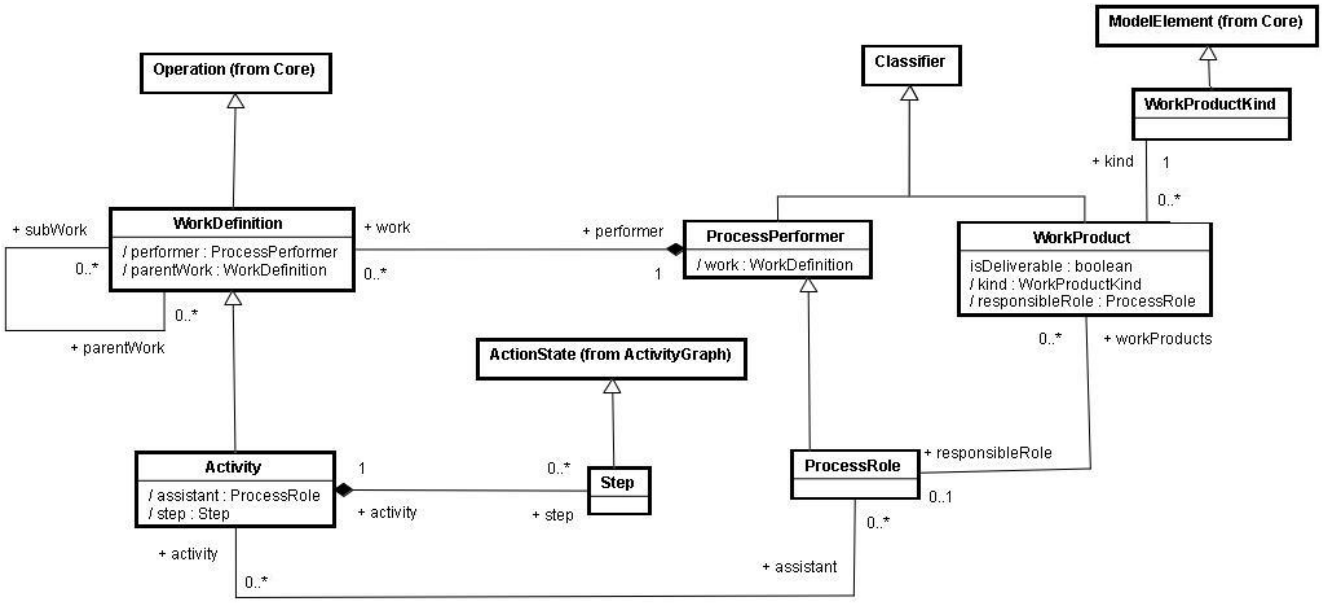

Figure 1: Process_Structure package of SPEM

for stressing SPEM's strengths and weaknesses because of its specific features—namely, its focus on modelling the social issues and the application environment, and its mechanisms for capturing the layered structure of complex systems.

So, this paper is structured as follows. Section 2 briefly presents SPEM and the main motivations for taking it as the natural reference for modelling SE methodologies, while Section 3 introduces the SODA methodology in short. Then, in Section 4 we outline the reasons for taking SODA as our case study, discuss how the SODA process could be modelled in SPEM, and highlight pros and cons of SPEM. Conclusions are reported in Section 5.

2. SOFTWARE PROCESS ENGINEERING META-MODEL

SPEM is an OMG standard object-oriented meta-model defined as an UML profile and used to describe a concrete software development process or a family of related software development processes. More concretely, as a meta-model SPEM allows for the formal definition of processes and their components.

The remainder of this section presents the main concepts of SPEM and compares SPEM with UML and OPEN, in order to show why SPEM is a natural candidate to model software development process.

2.1 Core Elements

SPEM is based on the idea that a software development process is a collaboration between active abstract entities called *roles* which perform operations called *activities* on concrete and real entities called *work products*. Each role interacts or collaborates by exchanging work products and triggering the execution of activities. The overall goal of a process is to bring a set of work products to a well-defined state.

Figure 1–from [26]– shows the main elements of the SPEM meta-model definition. SPEM introduces the concept of *WorkDefinition* that describes the work performed in a process. A *WorkDefinition* can be decomposed reflexively. Beside activities, there are other specialisations of *WorkDefinition* that are not drawn in Figure 1: *Lifecycle*, which is a sequence of *Phases*, and *Iteration*. These two elements describe/constrain the overall behavior of the performing process, and are used to assist with planning, executing and monitoring the process.

An *Activity* can be divided into atomic elements called *Steps*. A *Step* is a specialisation of *ActionState* (Figure 1), so that the flow of *Step* elements within an Activity can be represented by activity graphs [26, 27]. Each *WorkDefinition* is under the responsability of a unique role (*ProcessPerformer*). In the case of an *Activity*, a set of *ProcessRoles* can assist the main role for the realisation of the activity. SPEM also defines two further concepts: *Process* and *Discipline*. A *Process* corresponds to the root of a process model from which a tool can perform the transitive closure of a complete process. A *Discipline* allows activities within a process to be partitioned according to a common theme. Finally, *Guidance* elements may be associated with any other SPEM elements to provide more detailed information to practitioners about the associated element.

2.2 Why SPEM: UML and OPEN

The Unified Modeling Language (UML) is a language for specifying, visualising, constructing, and documenting the components of a software system [27]. As such, it has been used also to express meta-models of agent-oriented methodologies [4]. However, UML is not suitable to model the software development process: in order to obtain an effective representation of processes, a far richer set of concepts and symbols is needed.

To overcome these limits, UML *profiles* [27] can be used: these are UML extensions for building UML models related to specific domains. In particular, SPEM is a UML profile which extends UML expressiveness towards the modelling of software development processes. For instance, since a UML

profile can inherit UML diagrams from UML itself, it is possible to provide users with an easy-to-understand notation and exploit it for modelling software development processes. To this end, Use Case diagrams can be used to model actor activities, while Activity diagrams can be exploited to represent the flow of activities within a process. Moreover, a wide community of software developers is familiar with UML and uses a UML case environment tool. Defining a UML profile allows such a large community to reuse its knowledge and tools in the software-process modelling domain. On the other hand, SPEM makes it possible to overcome the limits of UML by adding all the concepts and symbols required to represent a software development process.

Besides SPEM, OPEN Process Framework (OPF) is another relevant process meta-model [24]. Both SPEM and OPF are conceived for an object-oriented context, so it is not surprising that they are based on the same basic conceptual model: there, a software development process is a collaboration between abstract active entities that perform operations on concrete and tangible entities [24, 26]. Moreover, both meta-models adopt UML as the main language for the description of processes and process components. There are, however important differences between the two approaches. First, OPF is not a UML profile: as such, it consists of a meta-model accompanied by a descriptive definition of its main ingredients. Instead, SPEM comes in the form of a UML profile, thus with a complete definition of the ingredients and a special purpose language notation which makes it easier to build comparable and reusable models with respect to OPF. In fact, SPEM provides a complete set of icons for the newly introduced concepts – the SPEM *notation* – that make it possible to build more comprehensible models, while OPF leaves the choice of a notation to the developer, though suggesting UML as a good candidate [24]. However, the actual use of SPEM is not straightforward, since the OMG proposal is quite generic and provides no clear directive about how to apply it. In fact, SPEM's semantics is essentially expressed in natural language, which could easily lead to imperfect process models due to the lack of a formal semantical definition for the SPEM concepts [6].

3. SODA: A SKETCH

SODA (Societies in Open and Distributed Agent spaces) [16, 17, 19, 1] is an agent-oriented methodology for the analysis and design of agent-based systems. Since the original version of 2001, SODA has always focussed on inter-agent issues, like the engineering of agents' societies and the environment for MASs: in this perspective, it has recently been re-formulated according to the Agents & Artifacts meta-model (A&A) [20, 22, 23], where artifacts take the form of computational devices that populate the agents' environment, and provide some kind of function or service used by agents [23]. In addition, SODA introduces a *layering principle* as an effective tool for scaling with the system complexity, applied throughout the SODA process. This layering principle consists of two mechanisms, *zoom* and *projection*: zoom makes it possible to pass from an abstract layer to another, while projection projects the entities of a layer untouched into another layer.

SODA is organised in two phases, each structured in two sub-phases: the *Analysis phase*, which is composed of the Requirements Analysis and the Analysis steps, and the *Designphase*, which is composed of the Architectural Design and the Detailed Design steps.

Requirement Analysis. Several abstract entities are introduced for requirement modelling. In particular, *requirement* and *actor* are used for modelling the customers' requirements and the requirement sources, respectively, while the *external-environment* notion is used as a container of the *legacy-systems* that represent the legacy resources of the environment. The relationships between requirements and legacy systems are then modelled in terms of suitable *relation* entities.

Analysis. The Analysis step expresses the requirement representation in terms of more concrete entities such as *tasks* and *functions*. Tasks are activities requiring one or more competences, while functions are reactive activities aimed at supporting tasks. The relations highlighted in the previous step are now the starting point for the definition of *dependencies* (interactions, constraints, etc.) among such abstract entities. The structure of the environment is also modelled in terms of *topologies*, i.e. topological constraints over the environment.

Architectural Design. The main goal of this stage is to assign responsibilities of achieving tasks to *roles*, and responsibilities of providing functions to *resources*. In order to attain one or more tasks, the role should be able to perform *actions*; analogously, the resource should be able to execute *operations* providing one or more functions. The dependencies identified in the previous phase become here *interactions*, i.e. "rules" enabling and bounding the entities' behaviour. Finally, the topology constraints lead to the definition of *workspaces*, i.e. conceptual places structuring the environment.

Detailed Design. The Detailed Design is expressed in terms of *agents*, agent *societies*, *artifacts* and artifact *aggregates*. More precisely, agents are intended as autonomous entities able to play several roles, while societies are defined as the abstractions responsible for a collection of agents. The resources identified in the previous step are now mapped onto suitable artifacts, while aggregates are defined as the abstractions responsible for a group of related artifacts. Finally, *workspaces* defined in the Architectural Design step take now the form of an open set of artifacts and agents.

4. SODA & SPEM

In this section, we first present the motivations that led us to take SODA as a case of study. Then, we present how we modelled SODA with SPEM: for space reasons, only a limited description of modelling is reported; readers interested in further details can refer to [18]. In the last part of this section, we discuss/report on the pros and cons of using SPEM to model SODA.

4.1 Why SODA

While software systems are growing more and more in complexity, object-oriented standard methods and techniques often fail to provide an adequate set of abstractions and mechanisms for engineering complex computational systems [10, 13]. With respect to other lines of research that are trying to address the same issues, like Aspect-Oriented Soft-

ware Development [10], Agent-Oriented Software Engineering (AOSE) [2] seems to be best suited to provide a coherent set of abstractions to manage the engineering of complex software systems. Among the many AOSE methodologies in the literature [14], SODA defines a process that focusses on fundamental issues such as *interaction* [21], *environment* [23], and *management of complexity* [17]. Therefore, putting SPEM to test by modelling the SODA agent-oriented process makes it possible to understand whether (and to which extent) the SPEM meta-model is expressive enough to capture the methodological abstractions and mechanisms that explicitly deal with complexity.

Interaction. Interaction in MAS can take two different forms: *social interaction*, involving agents interacting with each other, and *environmental interaction*, concerning agents' interaction with their environment. SODA is one of the few agent-oriented methodologies focussing on the modelling of such two types of interactions, since the erliest phases of the process.

Environment. Modelling the environment is a key feature for Engineering complex software system [28], and SODA actually considers this aspect a first-class issue—see Section 3.

Management of Complexity. Complexity frequently takes the form of a hierarchy [12], so that complex systems typically require layers in order to fully understand and reproduce their dynamics and behaviour. The software development process should then support some forms of system layering to support engineers in the design and development of this type of systems [9, 17]. Again, SODA deals well with this issue, too, by means of its *layering principle*—see Section 3.

4.2 Modelling SODA with SPEM

The first step to model a process with SPEM is to define the model root as an instance of *Process*—in this case, called SODA. The dynamic global behaviour of this process can be described by a suitable *Lifecycle* instance: in turn, this is further composed of two *Phase* instances – *Analysis* and *Design* – representing the homonymous SODA phases. Although the *Lifecycle* element is not mandatory – phases could be also associated directly to the process element–, its presence helps understanding the execution flow of the different phases in the process. *Phase* elements can be further decomposed into a set of *WorkDefinition* elements to model the sub-phases of the SODA process: Figure 2 shows the UML use-case diagram that models the relationships among the above *Phase* and *WorkDefinition* elements, using the *include* stereotype (*Lifecycle* is not shown). The decomposition is not over: in fact, each sub-phase can be decomposed in its turn into a set of correlated activities, modelled as *Activity* instances[18]: each activity will be associated to a responsible actor in the SODA analysis phase.

While *Lifecycle* expresses the dynamic part of the process model (that is, the process execution flow), *Discipline* instances are used to partition the activities of a process according to a common theme. In the case of SODA, we decided to group into the same *Discipline* the activities belonging to the same phase of the SODA development process, which led us to define two disciplines—*Discipline Analysis* and *Discipline Design*. Thus, each *Discipline* contains and

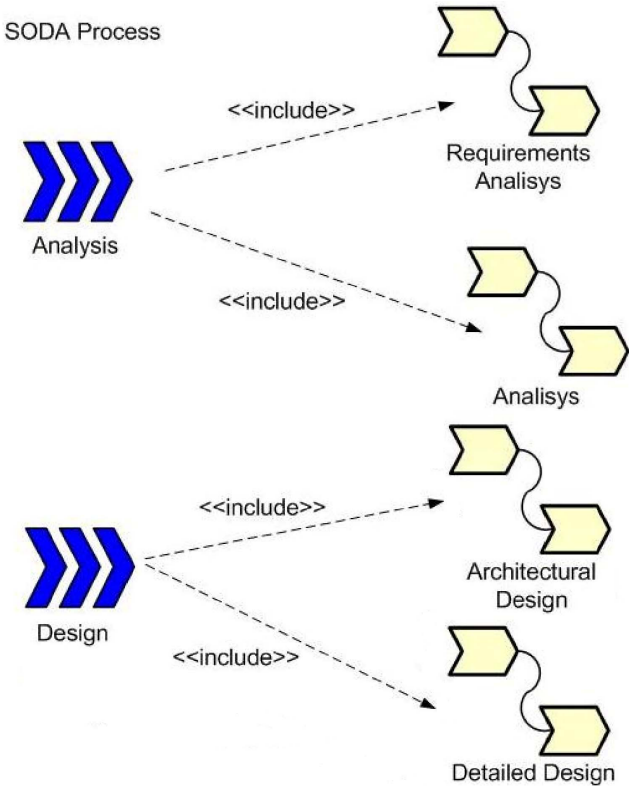

Figure 2: Use-case diagram of the SODA process: relationships between phases (left) and workdefinitions expressing sub-phases (right).

organises all the elements needed for modelling the corresponding SODA phase, including the diagrams that describe the execution flow of each activity.[1]

The SODA *Analysis phase* can be naturally characterised by three actors (*Roles*): one is responsible for the activities of *Requirements Analysis* sub-phase, another for the activities of the *Analysis* sub-phase, while the third is an application-domain expert aimed at assisting the two previous actors when analysing the application domain. These actors are modelled by three *ProcessRole* instances: *Requirement Analyst*, *System Analyst* and *Domain Expert*. Figure 3 shows the use-case diagram that models the relationships between the *Activity* instances and the *ProcessRole* instances in SODA's Analysis phase. Analogously, the *Design phase* is characterised by four actors: the *Architectural Designer*, responsible for the activities of the *Architectural Design* sub-phase; the *System Designer*, responsible for the activities of the *Detailed Design* sub-phase; and the aforementioned *System Analyst* and *Domain Expert*, which support the two previous actors in performing their tasks.

The execution flows of process activities are expressed via UML activity diagrams, which show the *WorkDefinitions*, the *Activities* and the involved actors, as well as the *WorkProduct* instances of each process activity. As an ex-

[1]Of course, in principle one could define a Discipline for each sub-phase instead that one for each phase. However, such a finer-grain approach seems unnecessary and inappropriate, since the activities of each phase are strictly related to each other.

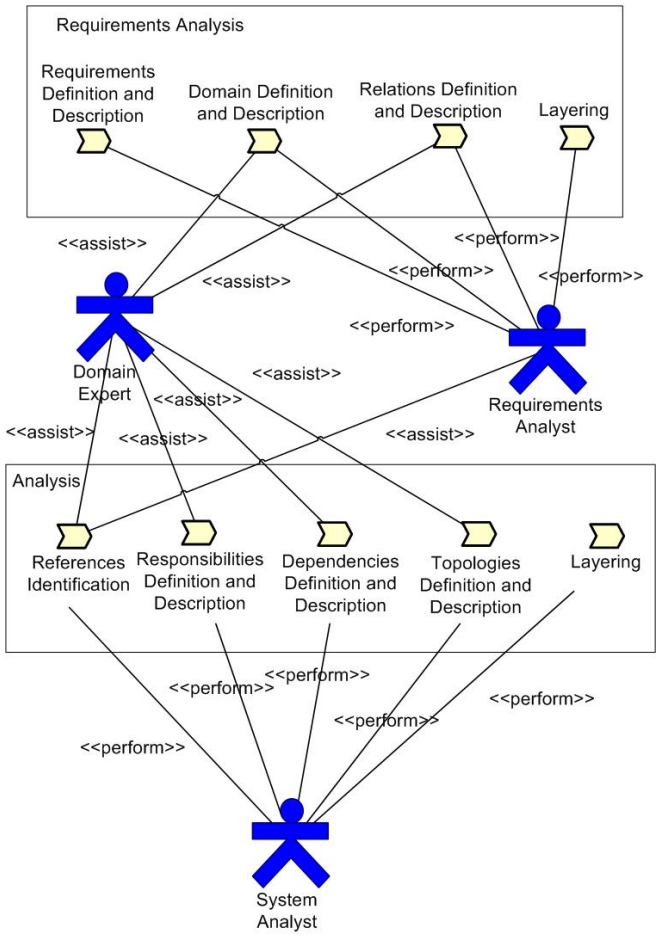

Figure 3: Use-case diagram of the Analysis phase.

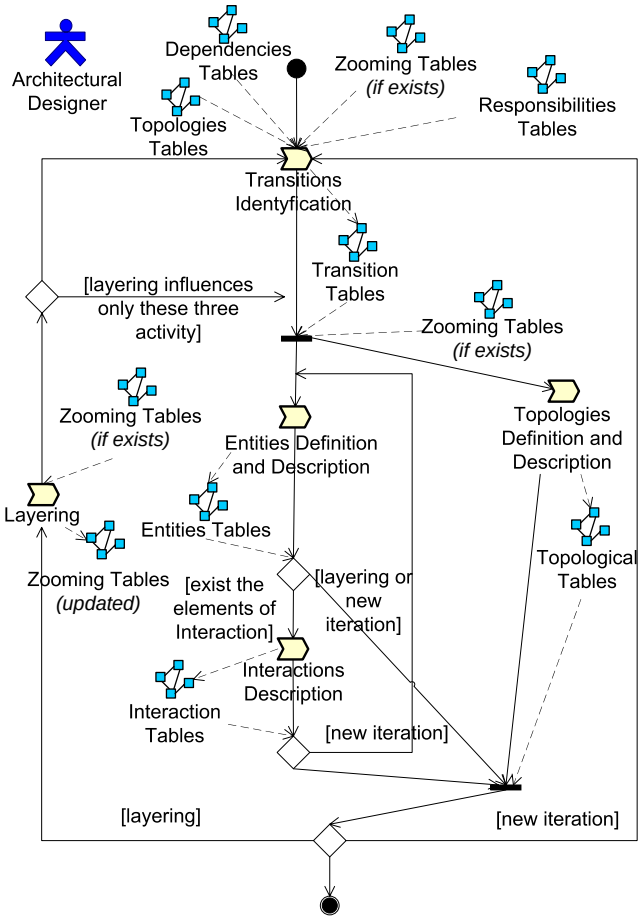

Figure 4: Activity diagram of the Architectural Design sub-phase.

ample, Figure 4 reports the Activity diagram of the activities performed in the *Architectural Design* sub-phase. The structure of *WorkProduct* instances and their relations with the SODA MAS meta-model [1] are then represented via UML Class diagrams (not shown).

4.3 Discussion

Despite its origin in the object-oriented context, SPEM could be applied to the agent-oriented SODA process quite naturally, yet with some limits in expressiveness and readability. On the one side, in fact, the software development process and its phases are similar in any methodology, and mostly independent of the computational paradigm adopted. On the other, however, agent-oriented methodologies introduce a richer set of abstractions and mechanisms, which naturally lead to define a more articulated software development process: this sometimes stresses SPEM to its limits, showing its weakness in facing the increasing complexity—in particular, UML diagrams often become nearly unreadable when applied to AOSE methodologies.

In the specific case of SODA, for instance, the key issues of interaction and environment (see Subsection 4.1) are apparently well modelled, instead the management of complexity presents some problems. In fact, process iterations and applications of the layering principle are not easily captured by Activity diagrams—see Figure 4. Moreover, the *WorkProduct* elements are characterised by a unique symbol, which makes it very difficult to express their change of state during process iteration. Currently, the only means provided by SPEM to face this issue is the *Guidance* element, which can be used to express these aspects by barely adding descriptions (comments) to the Activity diagrams (see Subsection 2.1): of course, an *ad hoc* diagram specifically designed for process modelling would be a much better solution. Use-case diagrams, too, become very complex when modelling actors' activities (Figure 4). In fact, SPEM provides just only one symbol to represent two different types of association, *perform* and *assist* [26]: so, it has to adopt different stereotypes – $\ll perform \gg$ and $\ll assist \gg$ – to tell one from the other. This choice, however, makes the diagram unreadable if there are many activities and roles: again, enhancing the language to provide different symbols for modelling such associations would be a more effective and expressive solution.

Other methodologies modelled by the FIPA Methodology Technical Committee [15] apparently do not suffer from such limitations, mainly because they do not include mechanisms similar to the SODA layering principle: so, in such cases SPEM turns out to be able to capture the whole methodology process in quite an easy way. For instance, iterations of the activities in ADELFE [3] can be modelled quite easily,

and also the activity diagrams obtained from this methodology are clear—in particular, they capture adequately the change of status of the ADELFE *WorkProducts*, while SODA diagrams are nearly unreadable due to the considerably larger number of *WorkProducts*. However, in the overall the SPEM notation is not expressive enough for correctly modelling all the dynamics of the AOSE methodologies: for instance, the change in the *WorkProduct* status above is represented only in the label associated to the *WorkProduct* itself—there is no standard way to do this inside the notation.

Summing up, SPEM is apparently a good base to model agent-oriented software development processes, although the above limits make it difficult to represent the process effectively, leading to models that are sometimes uneasy to understand. So, an extension seems necessary in order to enable SPEM to overcome its current limits in expressiveness and readability.

5. CONCLUSIONS AND FUTURE WORKS

SPEM seems to be the natural candidate for modelling Software Engineering processes—see Subsection 2.2. In this paper we apply SPEM in the context of AOSE methodologies, so as to test its general applicability by exploiting it in a different context with respect to the object-oriented field where it was initially conceived. Our experience with the SODA case study (see Subsection 4.1) shows that SPEM can be a good base meta-model for modelling processes defined by agent-oriented methodologies. However, some weaknesses and limits of expressiveness clearly emerge: as discussed in Subsection 4.3, UML diagrams often become unreadable, both for the intrinsic complexity of the agent-oriented methodologies, and for the lack of ad hoc entities that are necessary to obtain good methodology models. Such limits are intrinsic to the current SPEM definition, and could be overcome only by means of a suitable extension. For instance, some interesting extensions could be *(i)* a new Activity Diagram to capture and show the *WorkProduct* changes during iteration and layering; *(ii)* a new formalism to express guard conditions in the choice element, so as to enrich its semantics in capturing iteration and layering. In addition, as highlighted in Subsection 4.2, the introduction of new symbols and stereotypes could improve the readability of most diagrams. Finally, since AOSE methodologies often have to deal with many different kinds of *WorkProduct* and relations among *WorkProducts* and actors, one further addition to SPEM could be a mechanism for the management of the representation complexity similar to the SODA layering principle.

Therefore, in the future we also plan to test SPEM in other contexts, like for instance Aspect-Oriented Software Development [10], with the purpose of devising a possible SPEM extension that makes it general enough to potentially model any kind of process, independently of the computational paradigm adopted.

6. ACKNOWLEDGEMENT

This work has been supported by the MEnSA project funded by MIUR (Ministero dell'Università e della Ricerca) (PRIN 2006).

7. REFERENCES

[1] aliCE Research Group. SODA home page. http://soda.alice.unibo.it.

[2] F. Bergenti, M.-P. Gleizes, and F. Zambonelli, editors. *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*. Kluwer Academic Publishers, June 2004.

[3] C. Bernon, V. Camps, M.-P. Gleizes, and G. Picard. Engineering adaptive multi-agent systems: The ADELFE methodology. In B. Henderson-Sellers and P. Giorgini, editors, *Agent Oriented Methodologies*, chapter VII, pages 172–202. Idea Group Publishing, Hershey, PA, USA, June 2005.

[4] C. Bernon, M. Cossentino, M. P. Gleizes, P. Turci, and F. Zambonelli. A study of some multi-agent meta-models. In J. Odell, P. Giorgini, and J. P. Müller, editors, *Agent-Oriented Software Engineering V*, volume 3382 of *LNCS*, pages 62–77. Springer, 2004. 5th International Workshop, AOSE 2004, New York, NY, USA, July 19, 2004, Revised Selected Papers.

[5] S. Brinkkemper, K. Lyytinen, and R. Welke. *Method engineering: Principles of method construction and tool support*. Kluwer Academic Publishers, 1996.

[6] B. Combemale, X. Crégut, A. Caplain, and B. Coulette. Towards a rigorous process modeling with SPEM. In Y. Manolopoulos, J. Filipe, P. Constantopoulos, and J. Cordeiro, editors, *8th International Conference on Enterprise Information Systems: Databases and Information Systems Integration*, pages 530–533, 2006. ICEIS 2006, Paphos, Cyprus, 23-27 May 2006.

[7] M. Cossentino and V. Seidita. Composition of a new process to meet agile needs using method engineering. In R. Choren, A. F. Garcia, C. Lucena, and A. Romanovsky, editors, *Software Engineering for Multi-Agent Systems III*, volume 3390, pages 36–51. Springer, Feb. 2004.

[8] M. Cossentino and V. Seidita. Activity of the FIPA Methodology Technical Commitee. Technical report, ICAR-CNR, 2005.

[9] D. Dori. *Object-Process Methodology: A Holistic System Paradigm*. Springer, 2002.

[10] R. E. Filman, T. Elrad, S. Clarke, and M. Akşit, editors. *Aspect-Oriented Software Development*. Addison-Wesley, Boston, 2005.

[11] FIPA Methodologies. Home page. `http://www.pa.icar.cnr.it/~cossentino/FIPAmeth/`.

[12] M. J. Grene. Hierarchies in biology. *American Scientist*, 75:504–510, 1987.

[13] N. R. Jennings. Agent-Oriented Software Engineering. In F. J. Garijo and M. Boman, editors, *Multi-Agent Systems Engineering*, volume 1647 of *LNAI*, pages 1–7. Springer, 1999. 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'99), Valencia, Spain, 30 June – 2 July 1999. Proceedings.

[14] M. Luck, R. Ashri, and M. D'Inverno. *Agent-Based Software Development*. Agent-Oriented Systems. Artech House, Boston, MA, USA, 2004.

[15] Methodology Working Group. IEEE-FIPA methodology working group home page.

http://www.fipa.org/activities/methodology.html.

[16] A. Molesini, A. Omicini, E. Denti, and A. Ricci. SODA: A roadmap to artefacts. In O. Dikenelli, M.-P. Gleizes, and A. Ricci, editors, *Engineering Societies in the Agents World VI*, volume 3963 of *LNAI*, pages 49–62. Springer, June 2006. 6th International Workshop (ESAW 2005), Kuşadası, Aydın, Turkey, 26–28 Oct. 2005. Revised, Selected & Invited Papers.

[17] A. Molesini, A. Omicini, A. Ricci, and E. Denti. Zooming multi-agent systems. In J. P. Müller and F. Zambonelli, editors, *Agent-Oriented Software Engineering VI*, volume 3950 of *LNCS*, pages 81–93. Springer, 2006. 6th International Workshop (AOSE 2005), Utrecht, The Netherlands, 25–26 July 2005. Revised and Invited Papers.

[18] E. Nardini. Metodologie orientate agli agenti e standard SPEM: un caso di studio. Master's thesis, Bologna University, 2007.

[19] A. Omicini. SODA: Societies and infrastructures in the analysis and design of agent-based systems. In P. Ciancarini and M. J. Wooldridge, editors, *Agent-Oriented Software Engineering*, volume 1957 of *LNCS*, pages 185–193. Springer-Verlag, 2001. 1st International Workshop (AOSE 2000), Limerick, Ireland, 10 June 2000. Revised Papers.

[20] A. Omicini. Formal ReSpecT in the A&A perspective. *Electronic Notes in Theoretical Computer Sciences*, 175(2):97–117, June 2007. 5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'06), CONCUR'06, Bonn, Germany, 31 Aug. 2006. Post-proceedings.

[21] A. Omicini, S. Ossowski, and A. Ricci. Coordination infrastructures in the engineering of multiagent systems. In Bergenti et al. [2], chapter 14, pages 273–296.

[22] A. Omicini, A. Ricci, and M. Viroli. *Agens Faber*: Toward a theory of artefacts for MAS. *Electronic Notes in Theoretical Computer Sciences*, 150(3):21–36, 29 May 2006. 1st International Workshop "Coordination and Organization" (CoOrg 2005), COORDINATION 2005, Namur, Belgium, 22 Apr. 2005. Proceedings.

[23] A. Omicini, A. Ricci, and M. Viroli. Coordination artifacts as first-class abstractions for MAS engineering: State of the research. In A. F. Garcia, R. Choren, C. Lucena, P. Giorgini, T. Holvoet, and A. Romanovsky, editors, *Software Engineering for Multi-Agent Systems IV: Research Issues and Practical Applications*, volume 3914 of *LNAI*, pages 71–90. Springer, Apr. 2006. Invited Paper.

[24] OPEN. OPEN home page. `http://www.open.org.au/`.

[25] J. Ralyté and C. Rolland. An approach for method reengineering. In *Conceptual Modeling*, pages 471–484, London, UK, 2001. Springer-Verlag. 20th International Conference (ER 2001), Yokohama, Japan, 27-30 Nov. 2001. Proceedings.

[26] SPEM. SPEM Software Process Engineering Meta-Model home page. `http://www.omg.org/technology/documents/formal/spem.htm`.

[27] UML. Home page. `http://www.uml.org/`.

[28] D. Weyns, A. Omicini, and J. Odell. Environment as a first-class abstraction in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 14(1):5–30, Feb. 2007. Special Issue on Environments for Multi-agent Systems.