Advancing Object-Oriented Standards Toward Agent-Oriented Methodologies: SPEM 2.0 on SODA

Ambra Molesini*, Elena Nardini†, Enrico Denti* and Andrea Omicini†

*

Alma Mater Studiorum – Università di Bologna
Viale Risorgimento 2, 40136 Bologna, Italy
Email: {ambra.molesini, enrico.denti}@unibo.it

†

Alma Mater Studiorum – Università di Bologna a Cesena
Via Venezia 52, 47023 Cesena, Italy
Email: {elena.nardini, andrea.omicini}@unibo.it

***Abstract*—Building ad-hoc design processes and methodologies has become a key challenge in Software Engineering, and several efforts are being made for developing appropriate *meta-models* both for methodologies and development processes. The Software Process Engineering Meta-model (SPEM) – an OMG object-oriented standard – is a natural candidate for representing, comparing and reusing design processes in a uniform way.**

In this paper we apply SPEM 2.0 to Agent-Oriented Software Engineering methodologies, so as to assess its strengths and limitations. To this end, we take the SODA methodology as a significant case study, and compare the meta-model of its process obtained from SPEM 2.0 with the former meta-model obtained from SPEM 1.0.

I. Introduction

In the Software Engineering (SE) research field, several efforts are underway for developing appropriate *meta-models* for SE methodologies and processes. According to Cernuzzi et al. [1], a *Development Process* is an ordered set of steps that involve all the activities, constraints and resources required to produce a specific output which satisfies a set of input requirements. Typically, a process is composed of different stages/phases in relation with each other: each stage/phase identifies a portion of the work to be done, the resources to be exploited and the constraints to be obeyed for that purpose.

The relation between methodologies and processes is well studied in the literature: as pointed out in [1], methodologies focus more explicitly on *how* an activity or task should be performed in specific stages of the process, while processes may also cover more general management aspects about *who*, *when*, *how much*, etc.

Software development processes and methodologies have always been described in suitable terms for developers [2]: in fact, they talk about what tasks and techniques should be used, what sort of lifecycle is appropriate, and how these process elements should be organised in time and assigned to people. These aspects are often described in a manual or book that the project manager and his/her team of developers closely follow [2]. However, such manuals are not suitable for the automatic tools that typically support the designer's work, such as CASE tools that need specific rules for supporting methodologies and processes—rules stating, for instance, that it is a nonsense to put in a sequence two activities, three techniques and four roles: these rules are usually captured by a *meta-model*.

Although it is possible to describe a methodology / process without an explicit meta-model, formalising their underpinning ideas is valuable for checking consistency, or when planning extensions or modifications: there, meta-models can be exploited to check both the software development process and the completeness and expressiveness of methodologies. More generally, the relevance of meta-model becomes clear when studying the completeness and the expressiveness of a methodology / process, and when comparing or integrating different methodologies / processes together. For these reasons, research efforts are being made to define unified meta-models, aimed at representing the existing methodologies and processes in a uniform way, so as to promote their mutual comparison, their composition and reuse—this area is sometimes referred to as *Method Engineering* [3], [4].

SPEM (Software Process Engineering Meta-model, [5]) is one of the key references for this purpose: as it could be expected, SPEM is conceived for an object-oriented context, since most current methodologies adopt this paradigm as their reference.

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. Despite its origin in the object-oriented context, SPEM can be applied to the agent-oriented process quite naturally, since the process of software development is mostly independent of the computational paradigm adopted, and has essentially the same phases in any methodology. However, AOSE methodologies introduce a richer set of abstractions and mechanisms, which naturally lead to a more articulated definition of the software development process.

In a previous work [6] we explored the applicability of SPEM to the Agent-Oriented Software Engineering (AOSE) domain, whose abstractions and mechanisms are particularly suited to the design and development of complex software systems. There, we highlighted several limitations (briefly recalled in Section IV), exploiting the SODA methodology as a significant case study for stressing SPEM 1.0's strengths and weaknesses because of its focus on modelling the social issues and the application environment, and its mechanisms for capturing the layered structure of complex systems. Other AOSE methodologies modelled by SPEM [7] apparently do not suffer from such limitations (mainly because they do not include some mechanisms, like the SODA layering which is discussed below), so it seems quite difficult to determine general metrics and criteria for assessing the SPEM meta-modelling power.

So, in this paper we explore SPEM 2.0 by modelling the SODA methodology process and comparing the results with the previous ones—in particular, aiming to discover whether and how the previous limitations have been addressed: in a sense, to check whether the extension of the SPEM object-oriented standard has gone farther in addressing the many issues of agent-oriented methods and techniques.

Accordingly, the paper is structured as follows. Section II briefly presents SPEM 2.0 and some considerations about the adoption of SPEM in the AOSE field (Subsection II-A), while Section III presents the corresponding SODA process. Then, Section IV compares the meta-modelling power of SPEM 1.0 and SPEM 2.0, by taking the SODA process as its running example. Conclusions are reported in Section V.

II. SPEM

SPEM is an OMG standard meta-model for formally defining software and systems development processes [5]. The goal of SPEM 2.0 is not only to support the representation of one specific development process ore the maintenance of several unrelated processes, but to provide process engineers with mechanisms to consistently and effectively manage whole families of related processes promoting process reusability [5]. To this end, its meta-model introduces a clear separation between reusable methods content and its application in processes: the first item provides step-by-step explanations of how the development goals are achieved, independently of the placement of these steps within a development lifecycle; then, processes take these methods content elements and relate them into partially-ordered sequences that are customized to specific types of projects.

More in detail, SPEM 2.0 is structured into seven packages: *Core*, *Process Structure*, *Process Behaviour*, *Manage Content*, *Method Content*, *Process With Method*, and *Method Plugin*.

The *Core* package defines the base classes and abstractions for all other meta-model packages, while *Process Structure* provides the base for creating flexible process models — in particular, defining a process model as a breakdown or decomposition of nested *Activities*, with the related *Roles* and input / output *Work Products*. In addition, this package enables process reuse by providing mechanisms such as dynamic binding of *process patterns* (or *capability patterns*), which are reusable best practices for quickly creating new development processes.

The *Process Behavior* package supports the extension of the static breakdown structure of a process by externally-defined behaviour models. *Manage Content*, then, introduces the concepts to document and manage development processes through natural language description—indeed, practice of processes techniques and methods often cannot be formalised, but can only be expressed in natural language. In its turn, *Method Content* makes it possible to build a reusable development knowledge base which is independent of any specific development process: in particular, this package comprises textual step-by-step explanations, describing how specific fine-grain development goals are achieved by which roles, with which resources and results, independently of the placement of these steps within a specific development lifecycle. *Process With Method* provides what is needed to integrate processes with instances of Method Content concepts. Finally, the *Method Plugin* package introduces concepts for 'designing' and managing maintainable, large scale, reusable, and configurable libraries or repositories of method content and processes. In the next Subsection we outline some of the main issues in the use of SPEM in the AOSE context.

A. SPEM&AOSE

As introduced above, AOSE methodologies introduce a richer set of abstractions and mechanisms then OO systems, so the software development process is more articulated; in turn, the wide range of peculiarities of each methodology makes it difficult to define some general metrics and criteria for SPEM testing and evaluation.

Yet, some points can be put in evidence. First, each process and subprocess resulting from methodology representation should be reasonably clear and easy to understand, since failing to do so would make the SPEM representation itself little useful. SPEM's separation between Method Contents and Processes is a natural candidate to support this aspect, although the limited set of symbols offered by SPEM might lead to difficulties in representing elements or state changes. Second, since most methodologies exploit some iterative / incremental processes, the SPEM representation should be able to support such aspect. Third, since methodologies for complex systems typically include conceptual mechanisms for complexity management (such as some form of in/out zooming, the ability to view the system by levels at different abstraction levels, etc), some support should be provided by SPEM in order to capture such aspects in a satisfactory way.

In the following section we will try to exploit SODA as a testbed for evaluating SPEM 2.0's expressiveness with respect to such issues.

III. The SODA Process

SODA (Societies in Open and Distributed Agent spaces) [8], [9] is an agent-oriented methodology for the analysis and design of agent-based systems, which adopts the Agents & Artifacts meta-model (A&A) [10], and introduces a *layering principle* as an effective tool for scaling with the system complexity, applied throughout the analysis and design process.

SODA abstractions are logically divided into three categories: *i)* the abstractions for modelling/designing the system's active part (task, role, agent, etc.); *ii)* those for the reactive part (function, resource, artifact, etc.); and *iii)* those for interaction and organisational rules (relation, dependency, interaction, rule, etc.). In its turn, the SODA *process* is organised in two phases (Figure 1), each structured in two sub-phases: the *Analysis phase*, which includes the Requirements Analysis and the Analysis steps, and the *Design phase*, including the Architectural Design and the Detailed Design steps. Each sub-phase models (designs) the system exploiting a subset of the SODA abstractions: in particular, each subset always includes at least one abstraction for each of the above categories—that is, at least one abstraction for the system's active part, one for the reactive part, and another for interaction and organisational rules.

In order to represent the whole SODA process in a simple yet effective way, we exploited SPEM's separation between Method Contents and Processes (Section II): first, we modelled each sub-phase as a separate and independent Method Content, then we defined a specific process for each sub-phase – see Figures 3, 4, 5 and 6 for details – and re-used these processes to create the whole SODA process presented in Figure 1. In this way the whole process is reasonably easy to understand, since each sub-phase in the Activity Diagram is depicted as a simple activity, hiding the internal complexity of that process portion.

In addition, since the SODA process (Figure 1) is iterative and incremental, each step can be repeated several times, by suitably exploiting the layering principle: so, for instance, if, during the Analysis step, the System Analyst – one of the roles involved in the SODA process – recognises some omissions or lacks in the requirements' definition, he/she can restart the Requirements Analysis step adding a new layer in the system or selecting a specific layer and then refining it. Analogous considerations could be made for both the Architectural Design step – where the Analysis step can be restarted from the layering – and the Detailed Design step—which leads to restart the Architectural Design step.

The layering in Figure 1 is represented as a simply Activity of the process: actually, it is a *capability pattern* (Section II), i.e., a reusable portion of the process, as shown in Figure 2 where the layering process is detailed. In particular, the layering presents two different functionalities: *(i)* the selection of a specific layer for refining / completing the abstractions models in the methodology process, and *(ii)* the creation of a new layer in the system by in-zooming (i.e., increasing the system detail) or out-zooming (i.e., increasing the system

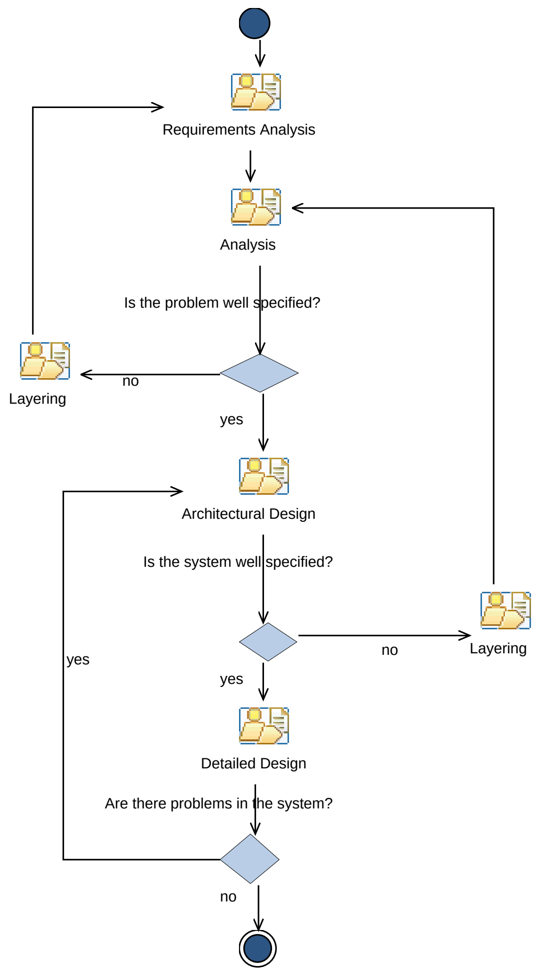

Fig. 1. Activity Diagrams of the whole SODA process.

abstraction) activities. In latter case, the layering process terminates with the projection activity needed to project the abstractions from one layer to another "as they are", so as to maintain the consistency in each layer.

The layering pattern is also used within sub-phases—except in the Detailed Design, where the layering principle is, by definition, not applicable. For instance, Figures 3, 4 and 5 report the sub-process of the Requirements Analysis, of Analysis and of Architectural Design steps, respectively: the layering activity is applied multiple times, both as a refinement or layer selecting technique in the single models (activities) – e.g., task layering, role layering, resource layering, space layering interaction layering, etc… – and as a way for re-starting the stage if some problems arise in the models or just for triggerring a new iteration of the stage. In the following, each sub-phase is presented in short.

a) Requirements Analysis.: Several abstract entities and models are introduced for this purpose. Each model is represented in Figure 3 as an activity, related to the corresponding Task in the Requirements Analysis Method Content. The latter specifies the steps to be completed to achieve the task, as well as the Workproducts to be produced—i.e., the SODA tables

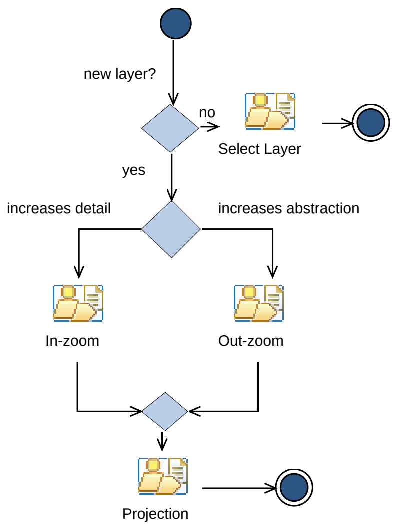

Fig. 2. Activity Diagram of the Layering Pattern.

describing the abstract entities of the Requirements Analysis. During the *Requirements modelling* activity (Figure 3), *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 in the *Environment modelling* activity. The relationships between requirements and legacy systems are modelled in the *Relation modelling* activity in terms of a suitable *relation*.

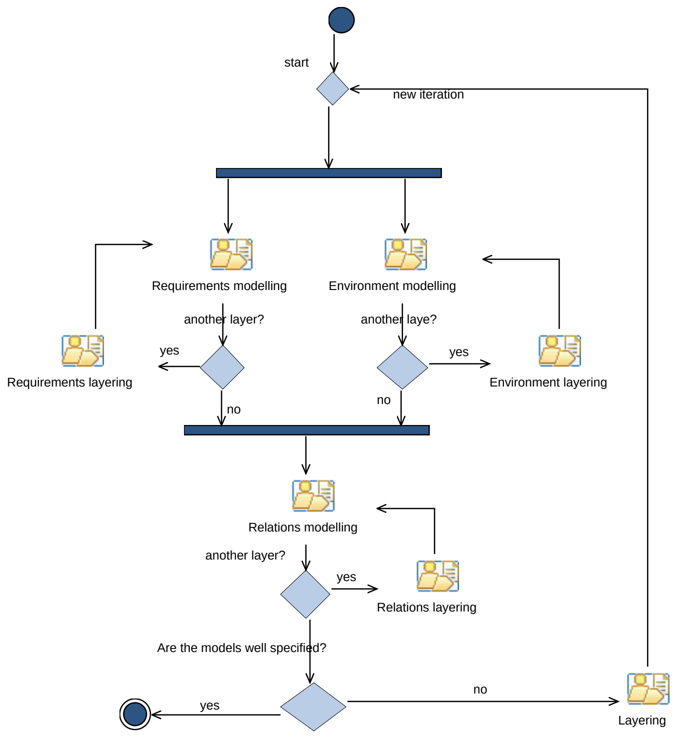

Fig. 3. Activity Diagram of the Requirements Analysis step.

b) Analysis.: The first activity in the Analysis step is *Moving from Requirements* (Figure 4), where the abstractions identified in the previous step are mapped onto the abstractions adopted in this stage to generate the initial version of the Analysis models. In particular, 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 and are analysed in the *Task analysis* activity, while functions are reactive activities aimed at supporting tasks analysed in the *Function analysis* activity. The structure of the environment, analysed in the *Topology analysis* activity, is also modelled in terms of *topologies*—i.e., topological constraints over the environment. The relations highlighted in the previous step are here the starting point for the definition of *dependencies* among such abstract entities in the *Dependency analysis* activity.

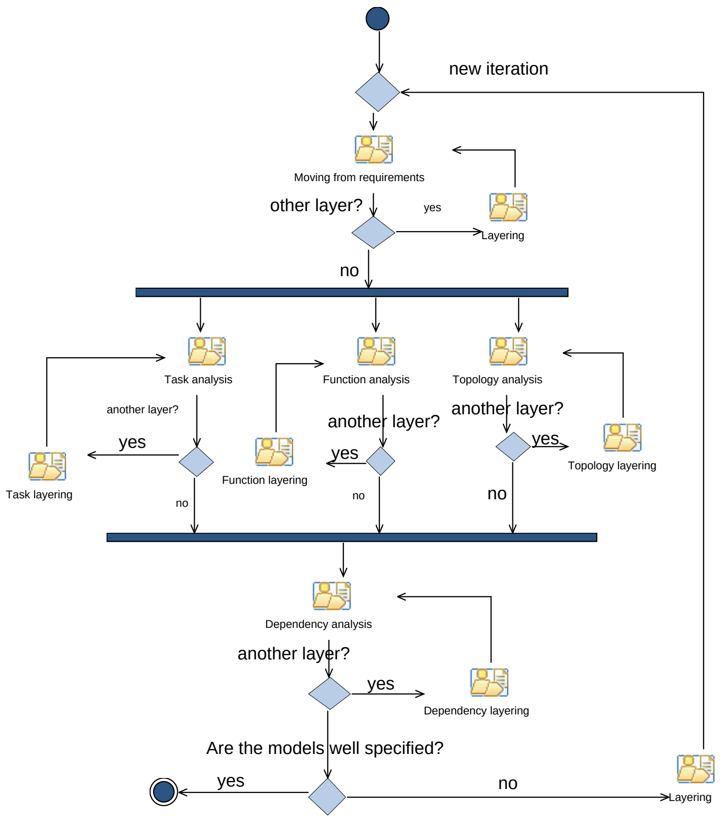

Fig. 4. Activity Diagram of the Analysis step.

c) Architectural Design.: This stage (Figure 5) is one of the more complex sub-phases in SODA. The first activity is *Transition* (Figure 5), where the abstractions identified in the previous step are mapped onto the abstractions adopted in this stage so as to generate the initial version of the Architectural Design models. The main goal is to assign responsibilities for achieving tasks to *roles* – *Role design* activity – and for providing functions to *resources*—*Resource design* activity. In order to attain one or more tasks, a role should be able to perform *actions* – *Role design* activity –; analogously, the resource should be able to execute *operations* providing one or more functions—*Resource design* activity. The topology constraints lead to the definition of *spaces*, i.e., conceptual places structuring the environment in the *Space design* activity. Finally, the dependencies identified in the previous phase become here *interactions* and *rules*. Interactions represent the acts of the interaction among roles, among resources and between roles and resources, and are designed in the *Interaction design* activity; rules, instead, enable and bound the entities' behaviour and are designed in the *Constraint design* activity.

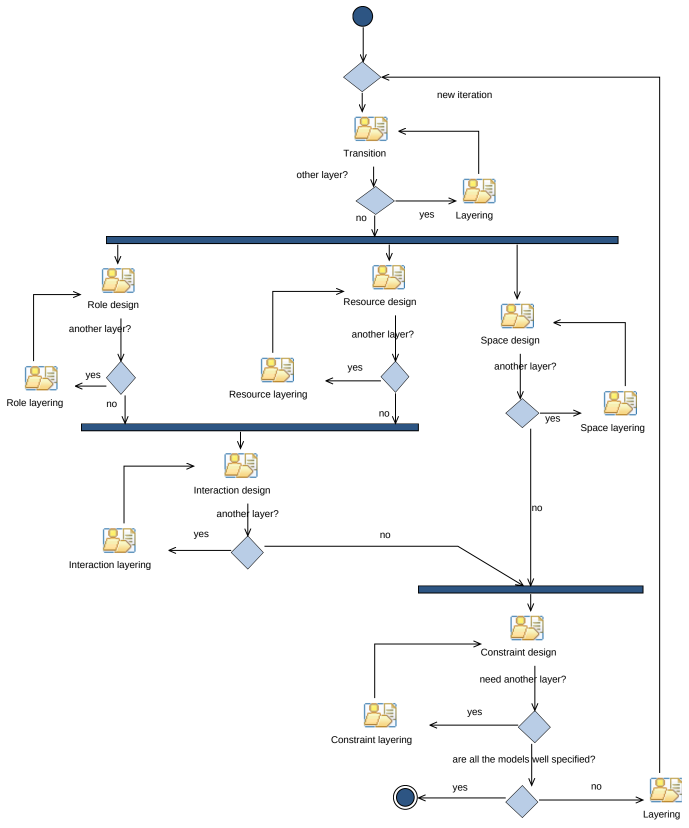

Fig. 5. Activity Diagram of the Architectural Design step.

d) Detailed Design.: The Detailed Design step (Figure 6) is the only stage where the layering principle is not applicable, since its goal is to choose the most adequate representation level for each architectural entity, thus leading to depict one (detailed) design from the several potential alternatives architectures outlined in the previous step. So, as shown in Figure 6, the first activity of this sub-process is *Carving*, which represents a sort of boundary between the Architectural Design and the Detailed Design, where the chosen system architecture is "carved out" from all the possible architectures. We also provide some SPEM's Guidelines for performing the carving activity properly. The next activity is *Mapping* (Figure 6), where the carved abstractions are mapped onto the abstractions adopted in this stage, thus generating the initial version of the Detailed Design models. These models are expressed in terms

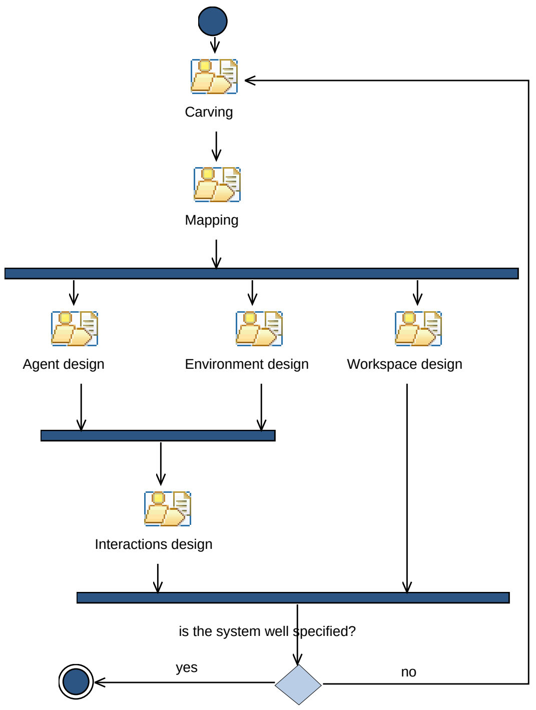

Fig. 6. Activity Diagram of the Detailed Design step.

of *agents*, agent *societies*, *composition*, *artifacts*, *aggregates* and *workspaces* for the abstract entities, while the interactions are expressed by means of *uses*, *manifests*, *speaks to* and *links to* concepts. More precisely, agents are intended here as autonomous entities able to play several roles, while a society can be seen as a group of interacting agents and artifacts whose overall behaviour is essentially autonomous and proactive: they are designed during the *Agent design* activity. The resources identified in the previous step are here mapped onto suitable artifacts, while aggregates are defined as a group of interacting agents and artifacts whose overall behaviour is essentially functional and reactive: they are designed during the *Environment design* activity. *Workspaces* take the form of an open set of artifacts and agents: artifacts can be dynamically added to or removed from workspaces, and agents can dynamically enter (join) or exit workspaces. Workspaces are designed in the *Workspace design* activity. Finally, the *uses*, *manifests*, *speaks to* and *links to* concepts are designed during the *Interactions design* activity.

IV. Discussion

In [6], the SPEM 1.0 meta-modelling power was put to test in the context of AOSE methodologies. There, SODA was taken as a case study to assess the strengths and limitations of SPEM, given its peculiar focus on the modelling and engineering (*i*) social issues, (*ii*) application environments, and (*iii*) complexity management—essential aspects for complex software systems. In order to simplify the comparison among the two versions of SPEM, Figure 7 reports the Activity Diagram of the Architectural Design stage as it was modelled in SPEM 1.0. Three major problems were put in evidence at that time:

1) Activity Diagrams and abstractions did not easily capture the SODA layering principle: this is quite clear in Figure 7, where layering is represented as a simply activity and there is no way to detail the layering sub-process without reporting in the Activity Diagram all the layering sub-activities;
2) WorkProduct elements are characterised by a unique symbol, which makes it difficult to model the state changes of a WorkProduct during the process evolution (Figure 7);
3) UML Diagrams often become unreadable due to the too many elements required to represent a process: for instance, Figure 7 shows how Activities, Roles, Inputs and Workproducts are depicted in the same diagram.

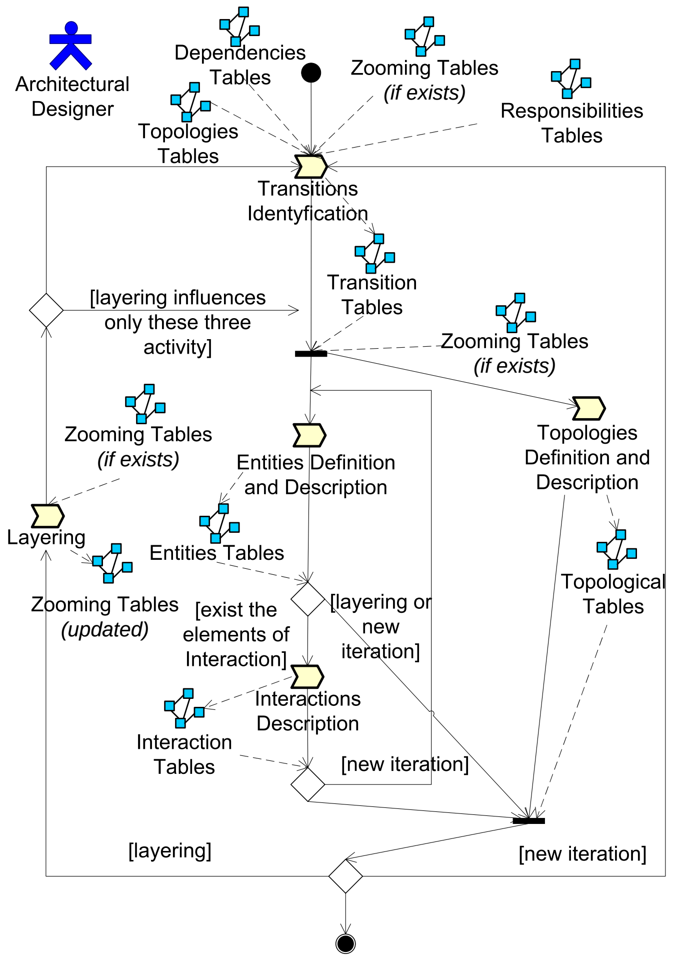
Architectural Designer
Dependencies Tables
Zooming Tables (if exists)
Responsibilities Tables
Topologies Tables
Transitions Identyfication
Transition Tables
[layering influences only these three activity]
Zooming Tables (if exists)
Zooming Tables (if exists)
Entities Definition and Description
Topologies Definition and Description
Layering
Entities Tables
Zooming Tables (updated)
[exist the elements of Interaction]
[layering or new iteration]
Topological Tables
Interactions Description
Interaction Tables
[new iteration]
[layering]
[new iteration]

Fig. 7. Activity Diagram of the Architectural Design step (SPEM 1.0).

These limits depend on the fact that SPEM 1.0 does not offer sufficient abstractions for effectively managing the representation complexity of articulated processes like those underpinned by SODA. From this viewpoint, SPEM 2.0 seems to overcome the limits of the previous version. In fact, the first issue is now addressed by providing the *capability pattern* mechanism (Section II) that makes it possible to represent a process pattern as a single activity, hiding its internal structure. As seen in Section III, such a pattern is suitable for modelling the layering principle, and allows engineers to realise more understandable and readable diagrams by hiding the process complexity behind the Activity abstraction. So, the different activities composing the Layering can now be detailed without reporting them in the Activity Diagrams each time, leading to a great simplification (compare Figures 5 and 7).

The second issue is addressed in SPEM 2.0 by extending both the UML Activity Diagrams so as to represent the input and output parameters of an Activity, and the UML State Diagrams so as to annotate the State elements [5]. Such extensions enable UML State Diagrams to model the lifecycle of each WorkProduct, and relate each State element to the corresponding Activity that causes the state change.[1] This makes it unnecessary to represent the Workproducts inside the Activities Diagrams as it was in SPEM 1.0.

The last issue is already partially addressed by the solution adopted for the first issue, since capability patterns simplify the Diagrams structure; in addition, as seen in Section II, SPEM 2.0 introduces the concept of *process reusability* and allows Method Contents to be defined independently of their application in the development lifecycle. So, Method Contents can be re-used by relating their elements into a process that is customised for the specific type of project. As a result, each UML Diagram is now more readable, as it can focus only on a given portion of the Method Content / Process, and does not contain all the “unusable” entities which are not related to the considered portion of the meta-model.

In Section III, for instance, we defined a Method Content for each SODA stage, relating them to the corresponding processes. The Method Content defines the involved Roles, the Tasks to be performed with the corresponding steps, the Inputs and Workproducts, and the relation between the Inputs / Workproducts and Tasks; processes, in their turn, specify the Activities responsible for the tasks achievement and their order inside processes. The resulting Activities Diagrams in SPEM 1.0 and SPEM 2.0 for the Architectural Design stage are shown in Figure 7 and 5, respectively: the latter appears more readable, as it does not contain the Roles and Workproducts that are not necessary in this Diagram.

Summing up, SPEM 2.0 seems to overcome the major limits of its previous version, providing the right abstractions and mechanisms to model articulated process like SODA’s, perhaps finding its way in the AOSE context.

V. Conclusions and Future Work

In this paper we took the SODA methodology as a case study for testing the applicability of SPEM 2.0 to AOSE methodologies. Moving from a previous work [6] where the SODA process was modelled in SPEM 1.0, we explored here whether SPEM 2.0 addressed the weaknesses and limits of expressiveness that had clearly emerged—mainly, the readability

[1]Example concerning WorkProduct elements are not reported here for obvious limitations in space.

of UML diagrams, both for the intrinsic complexity of Agent-Oriented methodologies, and for the lack of suitable ad-hoc entities.

Our experience indicates that SPEM 2.0 addresses such limits, by introducing a clear separation between Method Contents and Processes, adding capability patterns, and making it possible to express the ties between the Workproducts'states and the Activities that produce the changes in the Workproducts'states. Our next plans include testing SPEM in other contexts such as modelling the processes underpinned by MAS infrastructures, with the purpose of integrating AOSE methodologies and MAS infrastructures according to the Situational Method Engineering technique [11].

VI. Acknowledgements

This work has been supported by the MEnSA project (M*ethodologies for the* E*ngineering of complex software* S*ystems:* A*gent-based approach*) funded by the Italian Ministry of University and Research (MUR) in the context of the National Research 'PRIN 2006' call.

References

[1] L. Cernuzzi, M. Cossentino, and F. Zambonelli, "Process models for agent-based development," *Engineering Applications of Artificial Intelligence*, vol. 18, no. 2, pp. 205–222, March 2005.

[2] B. Henderson-Sellers and C. Gonzalez-Perez, "A comparison of four process metamodels and the creation of a new generic standard," *Information & Software Technology*, vol. 47, no. 1, pp. 49–65, 2005.

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

[4] J. Ralyté and C. Rolland, "An approach for method reengineering," in *Conceptual Modeling*. London, UK: Springer-Verlag, 2001, pp. 471–484, 20th International Conference (ER 2001), Yokohama, Japan, 27-30 Nov. 2001. Proceedings. [Online]. Available: http://www.springerlink.com/content/pbtbr52cwya7qyd4/

[5] Object Management Group, "Software & Systems Process Engineering Meta-Model Specification 2.0," `http://www.omg.org/spec/SPEM/2.0/PDF`, Apr. 2008.

[6] E. Nardini, A. Molesini, A. Omicini, and E. Denti, "SPEM on test: the SODA case study," in *23th ACM Symposium on Applied Computing (SAC 2008)*, R. L. Wainwright, H. M. Haddad, R. Menezes, and M. Viroli, Eds., vol. 1. Fortaleza, Ceará, Brazil: ACM, 16–20 Mar. 2008, pp. 700–706, special Track on Software Engineering. [Online]. Available: http://portal.acm.org/citation.cfm?id=1363686.1363853

[7] IEEE-FIPA Methodology Working Group, "Home page," `http://www.fipa.org/activities/methodology.html`. [Online]. Available: http://www.fipa.org/activities/methodology.html

[8] A. Molesini, A. Omicini, E. Denti, and A. Ricci, "SODA: A roadmap to artefacts," in *Engineering Societies in the Agents World VI*, ser. LNAI, O. Dikenelli, M.-P. Gleizes, and A. Ricci, Eds. Springer, Jun. 2006, vol. 3963, pp. 49–62, 6th Inter. Workshop (ESAW 2005), Kuşadası, Aydın, Turkey, 26–28 Oct. 2005. Revised Paper. [Online]. Available: http://www.springerlink.com/link.asp?id=j68l84713542525p

[9] SODA, "Home page," `http://soda.apice.unibo.it`. [Online]. Available: http://soda.apice.unibo.it

[10] A. Omicini, "Formal ReSpecT in the A&A perspective," *Electronic Notes in Theoretical Computer Sciences*, vol. 175, no. 2, pp. 97–117, Jun. 2007, 5th Inter. Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'06), CONCUR'06, Bonn, Germany, 31 Aug. 2006. Post-proceedings.

[11] M. Cossentino, S. Gaglio, N. Gaud, V. Hilaire, A. Koukam, and V. Seidita, "A MAS metamodel-driven approach to process composition," in *9th International Workshop on Agent Oriented Software Engineering (AOSE'08)*, M. Luck and J. Gómez-Sanz, Eds., AAMAS 2009, Estoril, Portugal, 12–13 May 2008.