An Expert System for the Design of Agents

Massimo Cossentino
SET - Université de Technologie Belfort-Montbéliard
Belfort cedex, France,
also with ICAR, Consiglio Nazionale delle Ricerche
Palermo, Italy
cossentino@pa.icar.cnr.it

Luca Sabatucci,
Valeria Seidita and
Salvatore Gaglio
Dipartimento di Ingegneria Informatica
University of Palermo, Italy
sabatucci,seidita@csai.unipa.it
gaglio@unipa.it

Abstract

The growing interest for the design and development of multi-agent systems has brought to the creation of a specific research area called Agent-Oriented Software Engineering (AOSE), specifically conceived for the development of complex systems. The development of such systems needs the support of appropriate tools that could help the designer in producing the design artefacts. We developed a tool called Metameth that may be used to define a new (agent-oriented) design process as well as to apply it. In this paper, we describe only a slice of this complex tool, specifically addressing the interaction with human actors (the designers). This subsystem is conceived as a collaborative multi-agent expert system, where each agent is capable of reasoning and adapting itself in order to support the designer in performing different kinds of design activities, regarding the use of various notations, and process life-cycles.

1. Introduction

In the last decade the need for a support during the development of more and more complex software has been partially fulfilled by a significant advancement in the application of software engineering. Moreover, the growing interest for the design and development of multi-agent systems, has brought to the creation of a specific branch called Agent-Oriented Software Engineering (AOSE) [24] [16] specifically conceived for dealing with the development of complex systems. Several design methodologies have been developed in the last years, each one with a specific attention to some categories of problems, development context or design approach. A great number of these methodologies are supported by CASE tools; in a not exhaustive list we can report Adelfe [5], Desire [1], INGENIAS [20], MaSE [12], PASSI [9] [7], Prometheus [19], and Tropos [2].
The positive effect of a CASE tool in the design phases is universally recognized, both in research and in industrial contexts. However we verified that a CASE tool has also negative effects on the evolution of a design process. This because even a small change in a part of of the methodology may require a new version of the tool, with a great effort spent in introducing new functionalities or modifying the existing ones. In other words, the effort for evolving a methodology is not proportional to the cost for modifying the correspondent tool. This is a real problem, especially in the research field, where the natural inclination towards experimentation and development of new ideas are hindered by the complexity of updating the software to new requirements.

We faced this problem by using the principles of Situational Method Engineering (SME) in order to find the proper way for building a new Software Engineering Process (SEP) [1]; in other words we aim at defining a Process for SEP Definition (PSEPD) with the specific goal of supporting the design of multi-agent systems.

In this paper we present the cooperation between the human designer and a part of the multi-agent system that we used to realize MetaMeth, the tool we developed in order to support our theories about Situational Method Engineering. The tool offers two main features: i) supporting the construction of a new process, and ii) supporting its enactment. Presenting the details of Metameth would require a lot of space, and for the sake of clarity, in this paper we only briefly introduce it; besides we prefer focusing the discussion on the subsystem devoted to provide the designer with a useful support features like automatic compilation of some artefacts, validation of the design and syntax checks. This multi-agent system is a collaborative expert system,

[1] other authors prefer terms like methodology, design process or simply process, in our work we use all of them as synonymous

capable of reasoning and adapting itself to the requirements coming from the new process definition phase; more in details, these agents support the designer in different kinds of activities, they understand different notations, and they can manage different process life-cycles.

The paper is organized as follows: section 2 describes our approach to the construction of a new design process, section 3 presents the MetaMeth tool, and its main functionalities. Section 4, details the implemented expert system and the agent society mediating between it and the tool users and, finally, some conclusions are given in section 5.

2. Building a New Software Engineering Process

In our work we have been focussing on the construction of ad-hoc design processes for developing multi agent systems. We found very promising the use of Situational Method Engineering and we are applying it to our research activity. Situational Method Engineering, provides means for constructing ad-hoc Software Engineering Processes (SEP) following an approach based on the reuse of process components [17] [13] [18] [4].

In Situational Method Engineering, a process is seen as composed of components (usually called textitmethod fragments or textitchuncks [3] [17]) regarded as elementary building blocks that can be extracted from existing design processes (or created from scratch), and then stored in a repository (sometimes called *method base*) from which they can be retrieved (during the *Method Fragment Selection* activity) for reuse in the new process assembly activity.

The approach we adopt for building a new process is reported in figure 1, for space constraints in the following of this section we will only give a brief description of this process, pointing out our attention to the contribution given by design tools (CAPE/CASE) during the process design activities; further details can be found in [8] [10]. During *Process Requirements Analysis* the designer (of the new process) achieves the necessary inputs for defining some fundamental requirements about the new process to be built. These requirements define the domain of interest this process should take into account. For instance, if the problem to be faced deals with transportation of human beings and someone in the development group has a good experience in applying formal methods then some safety properties of the system may be formally validated and the corresponding activities introduced in the new process.

A design process defines when and how someone does something in order to reach a specific objective [15], so the process requirements analysis should provide an initial list of the elements composing the new process. These elements can be: (i) activities (describing the work to be done), (ii) process roles (stakeholders performing the work), and (iii)

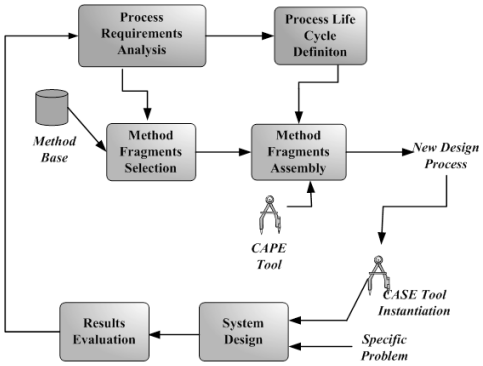

Figure 1. The adopted Process for Software Engineering Process Definition (PSEPD)

work products (artefacts resulting from some activities); we think that these elements can also be profitably used for the retrieval of method fragments (that will compose the new process) from a repository as discussed in [21].

This set of elements obviously affects the *Process Life Cycle Definition* activity where the designer makes decisions about the process model (or life-cycle) to be adopted in the new design process.

The *Method Fragments Assembly* activity results in the new process. This activity consists in putting together, following specific assembly techniques, the selected method fragments according to the structure prescribed by the identified process life cycle.

The definition of the new design process can be supported by a CAPE (Computer Aided Process Engineering) tool.

Finally, during the *System Design* activity, the MAS designer adopts the new process with the aid of a CASE tool (as it will be clear in the next sections this is instantiated by the previously cited CAPE tool). After that, the designed system is deployed and a *Results Evaluation* activity occurs in order to measure and evaluate the performance of the new process. Gathered information can be used in a further iteration of the construction process (if necessary).

3. Supporting the New Process Construction: The MetaMeth Tool

This section provides an overview of the Metameth tool, including an explanation of the requirements that lead us during its development and a brief description of its architecture. More details can be found in [10]. The development started with a requirement elicitation phase, driven by our previous experiences with the production and use of a couple of other tools for the PASSI design process. The first

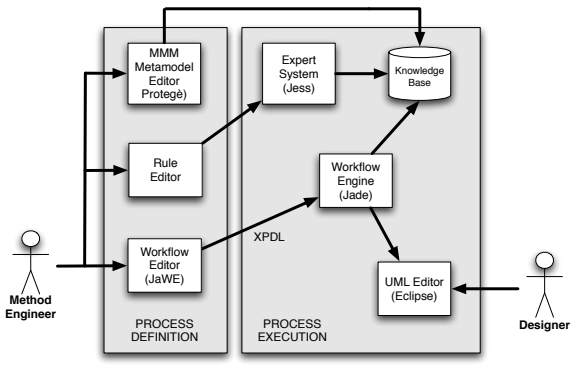

Figure 2. The architecture of MetaMeth

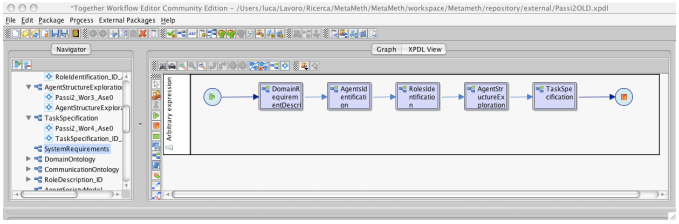

Figure 3. A screenshot of the JaWE tool used to describe a portion of a process.

one was the PASSI ToolKit (PTK), a plug-in for Rational Rose, which introduced several semantic checks on some aspects of the produced diagrams, the automatic composition of several artefacts, and the support of a design pattern repository. PTK has also been enriched with a code generation algorithm based on a multi-step transformation process.

The main limit of PTK is due to the direct integration of design rules in the source code, that led to a complex architecture, very difficult to be modified; as a consequence, while the PASSI process was evolving with improvements and new activities, the tool could not be easily updated.

The second experience in developing design tools was APTK (Agile PASSI Toolkit) [11], that is an add-in of a commercial design tool (Metaedit+ by Metacase [22]). APTK offers several features to the designer such as the automatic composition of some diagrams and patterns reuse support. The development of such a tool resulted in an interesting experiment, but with a high cost in terms of effort. The main lesson we learned from these experiences is the importance of separating the internal semantics of a methodology, from the graphic notation used to express it: a diagram is just a representation of the model according to some notation, not the model itself.

One of the major problems in applying the approach discussed in section 2, consists in ensuring the availability of adequate CAPE/CAME tools (for supporting the new process construction), and the development of a customised CASE tool for each new methodology.

The strategy we adopted for reducing the complexity of such a tool was to use a workflow-based approach as a core for process definition and execution: the employment of a couple of open source tools (JaWE and Shark by Enhydra [23]) provided several basic functionalities thus reducing the total required effort.

Basically, MetaMeth is structured in two macro-areas (Figure 2): process definition (corresponding to the functionalities offered by a CAPE tool) and process execution (corresponding to the application context of a CASE tool); these may be considered two separate systems able to interact because of the adoption of a common exchange data format (XPDL, the workflow specification language adopted by WFMC [6]).

The first system, concerning process definition (shown in figure 2) was mainly built over the JaWE component. The method engineer (the stakeholder devoted to construct the new process) is the main actor involved in working with this tool; he uses a graphical interface (see an example in figure 3) for defining the new process workflow, and for specifying additional information for each activity, such as the involved stakeholder and the user agent that will be responsible to manage it. Other useful functionalities offered by the Process Definition area are: i) ontology editor (here we use an existing tool: Protegé) and ii) rule editor. Together they enable the definition of ad-hoc procedures that will support the designer activities (as discussed in section 4).

The process execution part of the Metameth architecture (shown in Figure 2) is based on Shark [23], a workflow engine, that orchestrates the different design activities thus allowing their distribution and asynchronous collaborative execution. The global architecture of the process execution was conceived as a multi-agent system, where all the individual agents share a common knowledge about the object of the designer work (changing at run-time). Each designer involved in the project is essentially supported by a user-agent, which takes care of providing a set of functionalities conceived to support his activity. These functionalities are committed to a society of *Activity Agents*. A particular kind of Activity Agent is responsible for interacting with some external editors that are used by the designer to model the system. Actually, we have created a specific agent that works as an IBM Eclipse plug-in and supports the design of UML and PASSI-specific diagrams.

4. An Expert System for Supporting Design Process Activities

In this section we detail the part of MetaMeth conceived for supporting an intelligent interaction with the designer during the enactment of the process (CASE tool).

4.1. Motivation

While studying the intelligent part of our tool we focused our attention on the actions a designer performs while using a CASE tool in order to extract the sub-set of them that could be automated or anyway supported by the tool; results of this study are summarized in Table 1 [10]. The first category (GUI Actions) collects all the actions performed by the tool in order to support the design work of the user. These for instance include displaying forms where the user can introduce details of some elements; these forms can be composed of free text fields as well as lists of elements that have been already introduced in the design and can be reused in the specific context.
Just to provide an example, let us consider two PASSI activities [7]: *Agent Identification* and *Roles Identification*. Agents' names in the Role Identification activity (a sequence diagram used to represent agents' interactions) depend on those defined during the Agent Identification activity (delivering a use case diagram used to define agents and assign responsibilities to them). This kind of link between the two different activities of the design process can be supported by the automatic compilation of a list of elements from which the designer may select the items he wants to introduce in the current work.

Table 1. An overview of the operations that can be supported by a tool during the design process.

Action	Type of support
GUI Action	The tool interacts with the user (using a GUI) in order to support him in some operation.
WP Composition	The tool (partially) creates/updates a work product on the basis of the already introduced design information.
Rule Check	The tool grants a semantic/syntax check of the work product, alerting the designer to minor problems (warnings), major problems or conflicts (errors), and if possible, suggesting a solution.

Automatic composition of (part of) a work product is another way for a tool to support the design and development phases (WP Composition actions of Table 1). This scenario occurs, for instance, when two diagrams represent different views reporting the same information (from different points of view or with different levels of abstraction). Another application of the WP composition consists in the (automatic) production of documentation and source code for the system (since both documentation and source code are work products). Semantics that is behind diagrams may suggest an example for the third category of tool actions (Rule Check): referring again to the *Agent Identification* activity, we can say that the presence of a relationship between use cases assigned to different agents shows that those agents shall probably communicate (for instance for exchanging a service); if none interaction is depicted in the following Role Identification sequence diagrams, the system alerts the user to a possible error.

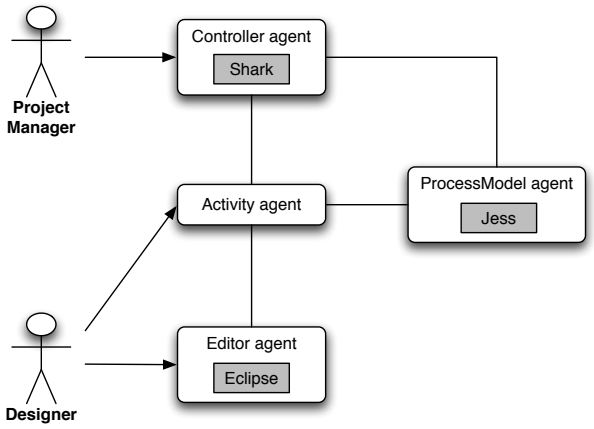

Figure 4. The agent society cooperating with the Metameth users.

4.2. Expert System Overview

Process enactment is supported by an agent society composed of four basic types of agents (shown in Figure 4):

- *A Controller agent*
- *A community of Activity agents*
- *A ProcessModel agent*
- *An Editor agent*

The Controller agent is responsible for the execution of the process: it activates the Enhydra Shark workflow engine and coordinates the execution of all the activities within the

instantiated process. The description of the process is provided to this agent by the process definition module in form of a XPDL file. The Controller agent interacts with the Project Manager that is in charge for assigning designers' roles (analyst, system architect,...) and for assigning rights to components of the design team. After this phase the Controller agent instantiates the design process. During the execution of the design process, the Project Manager can use this agent to access several information (list of completed activities, running activities and so on) and if the case he can also stop or terminate the process execution.

The ProcessModel agent is aware of the instantiated process and is responsible for managing the related design information. It owns a knowledge base where all the information about designed models is maintained, as a set of facts and rules. These elements are based on an ontology that describes all the concepts of the design process. Ontology concepts come from our definition of method fragment [8], and from the description of the multi-agent system in terms of its meta-model. As an instance we can consider the WorkProduct concept that is the result of an Activity (another concept of the ontology) of the process; a WorkProduct has a property called WorkProductKind representing the specific type of artefact (depending on the specific process, for example in PASSI we have Agent Identification, Role Identification and so on). Notation is another important property associated to a WorkProduct, it is used to define a graphic representation for the semantics that is behind an artefact.
The ProcessModel agent has reasoning capabilities (it uses a Jess module for that) and is able to perform some of the actions analyzed in the previous subsection (some others do not need the participation of the expert system to be realised). Actions performed by this agent are the consequence of the activation of Jess rules as it will be discussed in the next sub-section.
We can classify these rules according to five categories: i) validation rules, ii) semantic interpretation rules, iii) autocomposition rules, iv) update rules and v) import rules.
The first two categories of rules perform the Rule Check actions discussed in Table 1 while other rules realise the WP Composition actions of the same Table. The activation of a rule produces different results on the base of the specific category; as an instance, validation and semantic interpretation rules produce messages for the user; these can be Errors, Warnings and Messages. Other rules produce internal events thus triggering the execution of specific actions (such as a diagram auto-composition).
Errors and Warnings may be generated by syntactic and semantic validation rules; syntactic errors occur when the notation of an artefact is not respected (for example, in an UML class diagram it is not possible to draw a generalization relationship between a class and a package). Semantic coherence regards the elements of the MAS Meta Model of the methodology; for instance in a Role Identification diagram the designer cannot introduce an agent that has not been defined in the previous Agent Identification activity.
The Activity agent organization is conceived to interact with the designer by showing messages, proposing choices, and allowing the user to introduce data. When the Project Manager assigns an activity to a designer, the corresponding Activity agent is instantiated by the Controller agent; it moves to the designer location (if necessary) and then it asks to the designer if he accepts the new assignment. If the designer decides to accept, he can interact with the devoted Activity agent to start the activity and after completing that to mark it as done. The Activity agent is also responsible for contacting the Editor agent that opens (and manages) the diagram editor used for performing the activity. The Editor agent interacts with Eclipse in order to start the right plug-in and it also provides several user interfaces that realise the Gui Actions of Table 1. When the designer completes his work (or anyway saves it) this agent sends information about the designed artefact to the Activity agent using an XMI file. The Activity agent translates that to the RDF format (used in the ProcessModel agent by the ontology and expert system) and forwards the new file to the ProcessModel agent that in this way is updated about last design information.

Table 2. Rules defined for the two method fragments (AID and RID) of the PASSI methodology reported in the example in sub-section 4.1.

	AID	RID
Autocomposition	7	0
Import	24	27
Semantic Interpretation	10	10
Update	17	10
Validation	12	13
TOT.	*70*	*60*

4.3. Examples of Rules

As already discussed, we classify the rules processed by our expert system in five categories: i) validation rules, ii) semantic interpretation rules, iii) auto-composition rules, iv) update rules and v) import rules.

In the following we will provide two rule examples: one for syntactic validation and another for semantic interpretation.
The rule reported below performs a generic syntactic validation check that alerts a designer if he uses an incorrect element of the notation:

if

1. *a notation element (NE) exists*
2. *and NE is a notation element of NE-T kind*
3. *and NE belongs to a work product (WP)*
4. *and WP is of WP-K kind*
5. *and it does not exist a composition rule MME-NE-Link for WPK working on NE-T*

then

print a syntax error

The error message is shown to the designer because the specific kind of work product (WP-K) does not allow the presence of a notation element (of NE-T kind). The following code shows a portion of the corresponding Jess rule implementing the condition reported at the fifth point of the previous rule:

```
(defrule SYNTACTIC-VALIDATION::
         not-allowed-notation-element
"Warn if in a Work Product a not allowed
notation element is present"
.........
/* it does not exist a composition rule in
WPK specifying that a notation element
of NE-T kind is allowed*/
(not (MAIN::object
(is-a MMM-ElementNotationElementLink)
   (NE-Type ?NE-T)
      (MappingRuleOf $? ?WPK $?)
  )
)
=>  /* print an error message to
the standard error output stream */
(printout t "<Error>" crlf) (printout t "<![CDATA[")
(printout t "Syntax error, ")
(printout t "the work product " ?WP-NAME
                          " is of kind "
(slot-get (slot-get ?WPK DomainNamespace) Name)
"::" (slot-get ?WPK Name))
(printout t " so it can't contains " ?NE-T
"::" (slot-get ?NE Name)"]]>" crlf)
(printout t "</Error>" crlf))
```

This rule, working on generic elements of the ontology such as WorkProduct and WorkProductKind, can be applied to any kind of artefact, but it needs a specialization in order to fit a specific artefact. For instance, in order to create a validation rule for the Domain Description Diagram (DRD) of PASSI, it is necessary to add a pattern matching with the specific diagram kind:

```
/* WPK is a PASSI::DRD-Diagram kind*/
(test (and
(eq ?WPK-Name "DRD-Diagram")
(eq ?WPK-DN-Name "PASSI")
)
)
```

In this way it is possible to specify the real nature of an artefact by creating a binding with a specific validation rule for a DRD-Diagram of the PASSI methodology.

As anticipated, the second example regards a semantic interpretation rule; this is devoted to map the concepts of the adopted notation to those of the MAS meta-model (and viceversa); for instance let us suppose that in a PASSI Domain Requirements Description diagram (an use case diagram used to define system's requirements) there is a UML use case named *x*; the expert system knows, through the composition rules defined in the ontology for the specific work product, that a UML use case should be mapped (in the knowledge base) to a Requirement element of the PASSI metamodel; the consequence of the corresponding semantic interpretation rule is therefore that the expert system instantiates an entity Requirement named *x* in the MAS Model.

We will now detail the case of an element that is introduced for the first time in a specific diagram. The semantic interpretation rule is the following:

if

1. *a work product (WP) exists*
2. *WP is of kind WP-K*
3. *NE is of NE-T kind*
4. *NE-T kind in the context of WP-K is related to a MAS Meta Model element kind (MMMe-K)*
5. *a new element (NE) of MMMe-K is inserted in WP*
6. *NE has name NE-Name*

then

instantiate a new MAS Meta Model element of MMM-T kind, with NE-Name

Looking at the antecedent of this rule it is possible to note its generality: in fact there is no explicit reference to a specific diagram; the part of a JESS rule implementing the fourth item is reported below:

```
(defrule SEMANTIC-INTERPRETATION::
         general-semantic-interpretation-1
(MAIN::object
(is-a MMM-ElementNotationElementLink)
    (NE-Type ?NE-T)
(NE-Stereotype ?NE-ST-Rule &:(or
   (eq ?NE-ST ?NE-ST-Rule)
(eq ?NE-ST-Rule "$Any$")
)
)
(MappingRuleOf $? ?WPK $?)
(MMME-Type ?MMME-T)
(MME-DefinedHere ?WPK))
(test (not (eq ?MMME-T nil)))
=>
(printout t "<Message>" crlf)
(printout t "<![CDATA[")
(printout t "The notation Element " ?NE-T
            "::" ?NE-Name " has been mapped in the
MAS-Model as " ?MMME-T "::" ?NE-Name "]]>" crlf)
(printout t "</Message>" crlf)
(modify-instance (instance-name ?NE)
(Represented-MMM-Element (make-instance of
?MMME-T (Name ?NE-Name) (MAS-Model ?MM) [map])))
```

The previous rule presents a generic structure; in our work we have defined all of these rules with a generic approach depending only on the ontological representation of the process elements, this is general enough and independent from the specific design process; as a consequence, our rules can be applied to every kind of work product in the ontology. Despite of our efforts, for only a few rules, for instance

some semantic interpretation and auto-composition rules, we could not establish a totally generic structure; some of them are too much dependent on the specific design process and the specific work product to be completely generalized. These special cases require the introduction of new rules in the expert system at the time of the introduction of a new method fragment in the process under construction. Table II reports the rules we produced for the two method fragments examined in the two previous examples (Agent Identification and Role Identification fragments). It is interesting to note the presence of several rules for automatically composing the Agent Identification diagram. This is in fact an use case diagram reporting (at the beginning) the same information of the previous Domain Requirements Description diagram. Starting from that, the designer can identify agents by clustering use cases in packages that will represent each agent by collecting its responsibilities. Conversely no automatic composition work is provided for diagrams of the other method fragment; they are sequence diagrams used to depict scenarios and no automation is possible for them. As regards the other types of rules, as it is possible to see, there is no significant difference in the number of rules defined for the two fragments per category. This happens for the majority of fragments although there is a loose dependency on the amount of notation elements used in the diagram in some cases. Similar results we obtained for the other method fragments that are not here reported for length constraints.

5. Conclusions and Future Works

Several tools for supporting the development of a multi-agent system are discussed in literature, each of them is usually associated to a specific design methodology and provides different kinds of functionalities; some of them, for instance, provide graphical aids, some others verify the proper use of the notation and/or support the automatic generation of code. In the past we had some experiences in the production of similar tools; more specifically we developed PTk and APTk, respectively supporting the PASSI and Agile PASSI methodologies.

In this paper we presented a portion of the tool we developed to support our experiments of methodologies composition and, more specifically, we here focus our attention on the cooperation offered by an agent society (also employing an expert system) to the designer during the application of a design process; this system is able to perform syntactic validation, semantic validation/interpretation and auto-composition activities on the produced artefacts thus reducing the design time and lowering the risk of errors. Besides we also wanted our tool to be general enough for being applied to every methodology, this requirement was (almost totally) satisfied by the generality of the rules we produced for our expert system which knowledge base contains the description of both the used process model and the designed MAS model. However, some rules (for instance those related to semantic understanding of diagrams) remain in someway context specific and need a customization before being applied in a new diagram. Actually we are now working to introduce in the Metameth repository all the fragments discussed in [21]. In the future we will provide the tool with more features such as the possibility of interfacing it with an agent-oriented pattern reuse tool that allows code generation for one of the most diffused agent development platforms (Jade). The production of an extensive and well-formatted documentation of design artefacts is also scheduled and will be obtained by adopting a transformational approach implemented by a devoted society of agents.

References

[1] F. M. T. Brazier, B. M. Dunin-Keplicz, N. R. Jennings, and J. Treur. DESIRE: Modelling multi-agent systems in a compositional formal framework. *Int Journal of Cooperative Information Systems*, 6(1):67–94, 1997.

[2] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An agent-oriented software development methodology. *Autonomous Agent and Multi-Agent Systems (8)*, 3:203–236, 2004.

[3] S. Brinkkemper. Method engineering: engineering the information systems development methods and tools. *Information and Software Technology*, 37(11), 1995.

[4] S. Brinkkemper, R. Welke, and K. Lyytinen. *Method Engineering: Principles of Method Construction and Tool Support*. Springer, 1996.

[5] D. Capera, J.-P. George, M.-P. Gleizes, and P. Glize. The amas theory for complex problem solving based on self-organizing cooperative agents. In *Proc. of the 1st International Workshop on Theory And Practice of Open Computational Systems (TAPOCS03@WETICE 2003)*, pages 383–388, Linz, Austria, June 2003.

[6] T. W. M. Coalition. http://www.wfmc.org/.

[7] M. Cossentino. From requirements to code with the PASSI methodology. In *Agent Oriented Methodologies* [14], chapter IV, pages 79–106.

[8] M. Cossentino, S. Gaglio, A. Garro, and V. Seidita. Method fragments for agent design methodologies: from standardisation to research. *International Journal of Agent-Oriented Software Engineering (IJAOSE)*, 1(1):91–121, 2007.

[9] M. Cossentino and C. Potts. A case tool supported methodology for the design of multi-agent systems. In *The 2002 International Conference on Software Engineering Research and Practice*, Las Vegas (NV), USA, June 24-27 2002. ICSE '98, SERP'02.

[10] M. Cossentino, L. Sabatucci, V. Seidita, and S. Gaglio. An agent oriented tool for method engineering. *Proc. Of the Fourth European Workshop on Multi-Agent Systems. Lisbon, Portugal.*, 2006.

[11] M. Cossentino and V. Seidita. Composition of a New Process to Meet Agile Needs Using Method Engineering. *Software Engineering for Large Multi-Agent Systems*, 3:36–51, 2004.

[12] S. A. DeLoach, M. F. Wood, and C. H. Sparkman. Multiagent systems engineering. *International Journal on Software Engineering and Knowledge Engineering*, 11(3):231–258, 2001.

[13] B. Henderson-Sellers. Method engineering: Theory and practice. In *ISTA*, pages 13–23, 2006.

[14] B. Henderson-Sellers and P. Giorgini. *Agent Oriented Methodologies*. Idea Group Publishing, Hershey, PA, USA, June 2005.

[15] I. Jacobson, G. Booch, and J. Rumbaugh. *The unified software development process*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1999.

[16] N. R. Jennings. Agent-Oriented Software Engineering. In F. J. Garijo and M. Boman, editors, *Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World : Multi-Agent System Engineering (MAAMAW-99)*, volume 1647, pages 1–7. Springer-Verlag: Heidelberg, Germany, 30– 2 1999.

[17] K. Kumar and R. Welke. Methodology engineering: a proposal for situation-specific methodology construction. *Challenges and Strategies for Research in Systems Development*, pages 257–269, 1992.

[18] I. Mirbel and J. Ralyté. Situational method engineering: combining assembly-based and roadmap-driven approaches. *Requirements Engineering*, 11(1):58–78, 2006.

[19] L. Padgham and M. Winikof. Prometheus: A methodology for developing intelligent agents. In F. Giunchiglia, J. Odell, and G. Weiss, editors, *Agent-Oriented Software Engineering III*, volume 2585 of *LNCS*, pages 174–185. Springer, 2003. 3rd International Workshop (AOSE 2002), Bologna, Italy, 15 July 2002. Revised Papers and Invited Contributions.

[20] J. Pavòn, J. J. Gòmez-Sanz, and R. Fuentes. The INGENIAS methodology and tools. In *Agent Oriented Methodologies* [14], chapter IX, pages 236–276.

[21] V. Seidita, M. Cossentino, and S. Gaglio. A repository of fragments for agent systems design. *Proc. Of the Workshop on Objects and Agents (WOA06)*, 2006.

[22] J.-P. Tolvanen and M. Rossi. Metaedit+: defining and using domain-specific modeling languages and code generators. In *OOPSLA '03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 92–93, New York, NY, USA, 2003. ACM Press.

[23] T. E. Website. http://www.enhydra.org/.

[24] M. Wooldridge. Agent-based software engineering. *IEE Proceedings Software Engineering*, 144(1):26–37, 1997.