A Collaborative Tool for Designing and Enacting Design Processes

M. Cossentino
ICAR-CNR, Consiglio Nazionale delle Ricerche
Palermo, Italy
cossentino@pa.icar.cnr.it

L. Sabatucci
Dip. Ingegneria Informatica, Universitá degli Studi di Palermo
Palermo, Italy
sabatucci@dinfo.unipa.it

V. Seidita
Dip. Ingegneria Informatica, Universitá degli Studi di Palermo
Palermo, Italy
seidita@dinfo.unipa.it

ABSTRACT

Today several approaches using Situational Method Engineering paradigm exist, each of them proposes methods and techniques for developing ad-hoc design processes. In this context heavy efforts were spent in the construction of appropriate tools that could help method engineers in producing a specific design process and in using it. We developed a tool called Metameth for supporting the design process definition and its enactment. Metameth is implemented as a multi-agent system, where each agent is capable of reasoning and adapting itself in order to support the designer in performing different kinds of design activities.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Computer-aided software engineering (CASE)*; D.2.10 [**Software Engineering**]: Design—*Methodologies*

General Terms

Design Process, Supporting tool.

Keywords

Situational Method Engineering, Metameth, expert system, collaborative tool, CAPE/CAME/CASE tool.

1. INTRODUCTION

According to the Situational Method Engineering (SME) paradigm [3, 10], design processes are built by assembling reused parts of design processes called (method) fragments or chunks. Every kind of system development is usually aided by tools and several different tools are used in Software Engineering; in the same way a SME process requires the support of adequate tools for the definition of the process life-cycle, the reuse of fragments and the management of design process activities (Computer Aided Process Engineering-CAPE/Computer Aided Method Engineering-CAME tools) and for aiding the designer during the process development (CASE tools).

CASE tools are devoted to help the designer in the software development process by providing a software support for a reliable development of activities (like drawing diagrams, documenting artefacts, and so on), thus lowering the risk of errors and enhancing productivity.

Today, in the field of aided software development, MetaCase tools achieved large significance, they are able to provide an automated or semi-automated process for the creation of CASE tools, basing on a metamodel used to describe the language, the concepts and the relationships of a specific methodology.

In the field of Method Engineering a MetaCase tool is used to describe and to represent a set of methods; such a tool is sometimes called CAME tool. Several existing CAME tools (Mentor, Decamerone, MetaEdit+ and MethodBase) [13, 10, 2, 11] are also based on MetaCase technology allowing the construction or the automatically generation of CASE tools.

Moving the focus from methods to the whole process, we can note that Process Engineering shares the same targets with Method Engineering, they work on the same domain, but with different aims; Process Engineering mainly aims at defining processes rather than at modeling their methods (scope of interest of Method Engineering); both use the acquired knowledge on existing processes in the same way.

As CAME tools help the method engineer in the creation and definition of methods and in performing, constructing and representing them so CAPE tools allow the realisation of a process model. However a CAPE tool is more complex and sophisticated than a CAME because it should even offer a continuous supervising of the activities performed in the process.

In some previous works [5, 6] it was illustrated the way in which a Software Engineering Process can be constructed by adopting (and eventually extending) the Situational Method Engineering paradigm and we also conducted some experiments towards the production of a CAPE tool that could be instantiated in a specific CASE tool.

We verified that a CASE tool has often negative effects on the evolution of a design process because even a small change in a part of the newly design process, often requires a new

SAC'09 March 8-12, 2009, Honolulu, Hawaii, U.S.A.

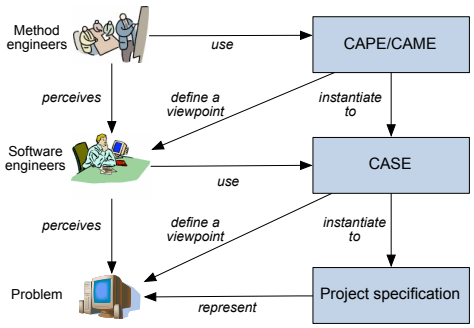

Figure 1: SME and Design Process Development (re-drawn from [14])

version of the tool, with a great effort spent in introducing new functionalities or modifying the existing ones.

In this paper we present Metameth, the tool we developed for supporting the design process definition and its enactment; the main contribution of this tool is that it is contemporaneously a CAPE and a CASE tool in the sense that once a new design process is created with the aid of the CAPE tool this latter can be instantiated in the specific CASE tool also able to offer intelligent support for the design phases.

The intelligent support provides features like automatic compilation of some artefacts, validation of the design and syntax checks.

We will detail all the techniques we used for making Metameth a collaborative tool during the design phase; in section 2 we will describe the whole Metameth project presenting its main functionalities and main components also describing the expert system that is one of the core components of the tool. Section 3 introduces a case study that derives from the application of the whole approach. In Section 4 some related works are illustrated and finally some conclusions are given in Section 5.

2. METAMETH PROJECT OVERVIEW

The construction of ad-hoc design processes with the SME approach involves a specific stakeholder, the method engineer (we already referred to him as process designer), whose tasks are to define and describe the new process in a way that could allow new system designers (users of the newly designed process) to use it and to solve a specific problem [14, 3](see Figure 1). The new system designers' work is supported by tools (CASE tools) for producing system specifications, in the same way the method engineer's work is supported by tools.

The CAME tools supply aid in method fragments construction and definition and in their integration; in our work we also needed a CAPE for defining the whole process (not merely method fragments) and for managing process activities. Once the design process is created this tool should be instantiated in a CASE tool that is customized for the specific process.

Metameth allows the composition of different processes starting from a repository of fragments and the related notations that were defined in [12, 5].

Each fragment, we retrieve from the repository, is composed of a workflow that structures all the activity to be done for producing a work product, each work product is devoted to define, refine or quote at least an element of the metamodel; the metamodel gives to the process the awareness on the kind of contribution each of its element produces.

All the previous elements have an effective "implementative" counterpart that is: an *XPDL file* for implementing the workflow part of the fragment, a *set of design rules* that orchestrates the composition rule for each work product, an *Activity agent* that interacts with the designer in order to support him during the design phase and finally a *set of format transformations* supporting the role of the Activity agent when it helps the designer in interfacing with a specific design tool (see later section 2.2).

In the past two experiments were made: the creation of a CASE tool as plug-in for Rational Rose [7] that featured several checks in the design semantics and the automatic composition of several design diagrams. Code generation was also supported by a pattern reuse tool and good results had been obtained about that. The second was a situational method engineering experiment [9] whose result was an agile process (Agile PASSI) and the related CASE tool based on MetaEdit+ (by the MetaCase company) [15]; MetaEdit+ flexibility in modeling the semantic and notation of single method fragment proved to be very useful. We extended MetaEdit+ features by including the automatic composition of some diagrams (mostly based on reverse engineering of code for documentation purpose) and pattern reuse support. The development of such a tool although simple and not totally engineered had a high cost and was not easily reproducible for future experiments of design process construction above all because of the great dependency the CASE tool presents towards the design process it is supporting.

From this experiences we deducted the need for the support of a more flexible tool that could easily help in building whatever process we would decide to build. For this reason we decide to start the construction of a new tool (Metameth) satisfying the needs for automatic support for fragments repository maintenance, process composition, and process enactment. One of the most relevant limits of our previous tools was the impossibility of easily sharing the different design phases in the design team (collaborative work was not supported) and the resulting artefacts (editable model files). To overcome this limitation we decided to explicitly support distributed design processes as one of our main goal. Being a method engineering tool it should have obviously supported several different design processes.

Finally, for a better support to the designer's work, the tool was designed for including the possibility of automatically composing (portion of) diagrams, performing syntax checks on notational aspects of the design and the consistency checks about some design issues like the correct instantiation of the most important elements of the system metamodel.

2.1 The Metameth Architecture

Figure 2 shows the architecture of Metameth by focusing on the flow of work performed in a SME process that can be inferred from Figure 1; we decided to separate the definition phase of the new processes from its enactment, this choice was a direct consequence of: *(i)* the clear separation between method engineer's work and designer's one and *(ii)*

the decision of using an approach based on the concept of workflow for the definition, and then the execution, of the process [1].

Metameth main sub-systems are reported in Figure 2 and a great part of them was not developed from scratch. The most relevant activity on them concerned their integration and the enablement of their information exchanges; the most relevant (open source) components we adopted are:
- **JaWE** and **Shark** (by Enhydra), **Jade** and **Jess** (a relevant portion of its services are required by the Activity Agents that need reasoning capabilities)
- The **Metamodel Editor** required by the adopted design process is depicted in form of an ontology by using the Protégé tool;
- **Rule Editor** that is the tool we built for describing Jess rules starting from a template;
- **Knowledge Base**. The Jess system operates on a knowledge base storing all the information about the design. This means that all instances of a specific design concept and all of its relationships with the other concepts will be stored. The concrete result of this approach is that the complete model of the designed system is stored in the knowledge base and can be accessed by the inference engine.
- **Design Model Editor**. Metameth ideally supports any design model editor with the unique limit that it has to import/export design models in a format that could be transformed in first order logic facts (used by the JESS inference engine). In the current status of the project we decided to adopt the Rational System Developer tool that supports the XMI exchange format.

Metametht main architectural components concur in supporting our approach as follows: once the method engineer (see Figure 2) has modelled the process [2] he can use a graphical tool (JaWE) to produce its XPDL translation (XPDL is the process specification language adopted by WFMC). The method engineer can specify, for each activity, additional information, such as the user that will be responsible to manage it and the involved stakeholders. The next step is the specification of the semantics of the metamodel elements (by using Protégé and the composition rules for work products delivered within the process (these rules are specified by using Jess rules).

Once the design process has been entirely described using the XPDL language (process aspects) and Jess rules (reproducing metamodel semantics and composition rules), the method engineer (hence the process administrator using workflow glossary) may instantiate it using the process execution module.

This module was developed using a multi-agent system in order to take profit of the possibilities offered by the Jade platform (agents mobility and semantic communications). Agents' mobility was an interesting feature for our purposes since it permitted us to reduce the complexity of client platforms. Each designer only needs to install a package containing the Jade platform (a Jar file) and an agent responsible to start the communication with the workflow engine (so that this latter is aware of the address in the network of each designer). Agents devoted to support the different activities will move to the designer's host when requested and therefore a dynamic configuration of the client platform is easily achieved. The main elements of the part of Metameth used for enacting the design process are:
- A **Controller agent** that is interfaced with the workflow execution engine and is responsible for the execution of the process (it loads and starts the process).
- One or more **Stakeholder agents**, one for each designer that collaborates in the process; the designer can accept, start, and decline the activities assigned to him during process definition. After a login session, the user may verify his activity list and can start/ refuse or delegate an activity.
- A **ProcessModel agent** used to wrap the Jess expert system that holds the knowledge base where the instantiated system model is stored.
- One or more **Activity agents** that interact with the designer by showing messages, proposing choices, and allowing the user to introduce data and open the diagram editor used for documenting the activity.

The ProcessModel agent is aware of the instantiated process and is responsible for managing the related design information; all the information about designed models are stored in a knowledge base in the form of a set of facts and rules. As already discussed, these elements are based on an ontology (metamodel) that describes all the concepts of the domain specific language.

As regard the flow of information, in this case the ProcessModel agent (see Figure 2) gets information about the current state of the process from the Knowledge base through a XMI file and translates them to a RDF file for communicating with the Activity agent. When the designer accepts to perform an activity, an Activity agent moves to his platform and it becomes responsible for coordinating all the operations related to the specific activity (also in collaboration with the ProcessModel agent and the design model editor).

The Activity agent generates and sends to the DesignModel editor the design file with a XMI file and can receive from it the project file after the designer has changed it.

Activity agents offer several services to the designer: *i)* auto-composition used when a work product can be automatically modified/created or updated; *ii)* notation interpretation, used to map notational elements (e.g. use cases, classes, activities, etc.) into elements of the metamodel, *iii)* semantic validation used to verify the semantic consistence of the whole project. This is done by communicating with the ProcessModel agent and exchanging Jess facts and RDF file for respectively describing the composed diagrams and updating the knowledge base.

The metamodel described using Protégé is inserted in the knowledge base by exchanging an RDF file between the two modules and the design rules are stored in the Rules database by sending to it a Jess Rules file from the Rule Editor.

Metameth intelligent part main task is to assist the designer during the design activities and to provide him with the right helps for lowering the risk of errors and for speeding up his work.

The construction of the expert system was based on the observation of the actions a designer performs during his work in order to establish which of them could be computerized and for which of them it could be provided aids in the form of messages and warnings. The set of identified actions for which support was provided is the following (more details can be found in [8]):

[1]For the definition of workflow as well for its description in terms of XPDL language, we referred to the WFMC specifications [1].
[2]This part of the work in not (still) integrated Metameth.

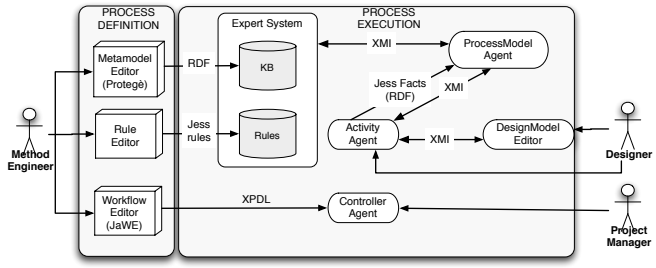

Figure 2: Metameth Overview

(i) **GUI action** - using a GUI the tool interacts with the designer and supports him during his work.
(ii) **WP composition** - the set of operations a designer performs while working on a specific work product; he can create or update (edit, modify) a work product and he could need information about the output produced by other artefacts. An interesting case occurs when the output of an activity produces an information that affects the status of a document produced in another activity; when this happens, the tool supports the automatic creation or update of the affected work product.
(iii) **Rule Check** - the tool supports the design through the generation of warning/error messages and eventually by proposing solutions after the execution of syntactic and semantic checks on work products.

Each of the above listed action category has a correspondence to a set of rules allowing the Jess module to perform the reasoning capabilities of the ProcessModel agent; we classified these rules in five categories: *validation rules* and *semantic interpretation rules* that perform the Rule Check actions, and *auto-composition rules*, *update rules* and *import rules* that perform the WP Composition actions. These rules are enabled by the representation in form of Jess rules of the system metamodel and its composition rules. It is therefore easy, to check the correctness of facts in the knowledge base (representing the instantiated system model) and to validate some properties of the produced work products.

2.2 Interaction with external tools

Metameth does not specify an unique tool for the construction of design artefacts as it happens in other approaches. For sure, binding Metameth to a specific editor could have simplified the development phase, but, on the other hand this would limit the possibility of supporting other notations. Therefore a generic approach was required for allowing the interaction with multiple editors. This was challenging to realize, because of the difficulty to manage different kind of external applications, each one with different principles and specific API to consider. In order to simplify this task, the possible interactions between Metameth and a generic external tools have been classified in:
- opening of the tool;
- creation of a new project/document;
- introduction of a custom widget (button, menu, or toolbar);
- population of an empty document with data from the already designer model (from the knowledge base);
- setting a specific layout for elements of the document;
- on application closure, transferring of document data to the Metameth internal model (in the knowledge base).

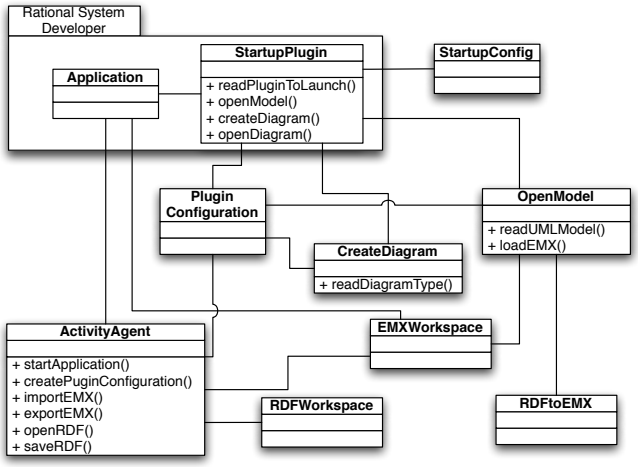

Figure 3: Metameth subsystem - The interaction with an external tool

Depending on the external tool, not all these features are available because of limitations given by the offered API. This section illustrates an example of interaction with an UML editor, the Rational System Developer by IBM. Figure 3 illustrates the subsystem aimed at providing the interaction with the Rational System Developer. The Activity Agent is responsible of controlling the external tool. Classes inside the package are specific of the Rational System Developer API that is used by Metameth. Two classes, *OpenModel* and *CreateDiagram*, are registered as plugins, and automatically start when the Rational System Developer is opened. They represent the bridge between the editor and Metameth, interpreting commands and operating data transformations. When a new diagram is created, the CreateDiagram class receives a command where the "Name" value defines the name of the artefact to be created and the "DiagramKind" value specifies the kind of diagram. The OpenModel class is responsible to generate a diagram from an existing model. It opens the model and populates the diagram, giving to elements the appropriate layout. The RDFtoEMX class is responsible of the transformation from the Metameth internal model (that is expressed in Jess facts that are exportable in RDF format), to the Rational System Developer model, that is stored in EMX, a language based on XMI. When the work is terminated, the inverse transformation, from EMX to RDF is operated directly by the ActivityAgent by an XSLT.

3. CASE STUDY

This section illustrates an example of Metameth use: the CASE tool instantiation for a specific design process. In this experiment we used Metameth for supporting an existing design process (PASSI [4]), we did not construct a new one by using SME, but this did not effect the relevance of the case study since the process is defined from scratch in Metameth just as it would happen for a totally new project.

The starting step is the definition of the PASSI process;

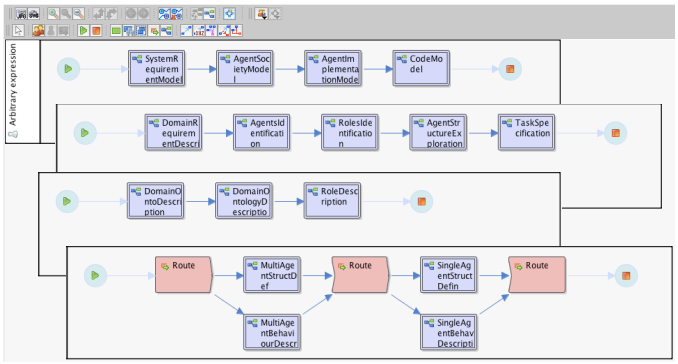

Figure 4: PASSI definition with JaWE.

Table 1: The rules instantiated for PASSI.

	Auto Composition	Semantic Interpretation	Semantic Validation	Syntactic Validation	Update	TOT.
DRD	0	2	3	3	3	**11**
AID	7	4	15	3	5	**34**
ASE	5	3	8	3	2	**5**
RID	0	4	11	3	2	**20**
TSP	1	4	7	3	3	**18**
DOD	0	6	10	4	1	**21**
COD	2	6	6	2	1	**17**
RD	5	7	5	4	1	**22**
MASD	4	4	5	3	1	**17**
MABD	2	1	7	6	1	**17**
SASD	4	4	3	3	1	**15**
SABD	6	0	4	3	1	**14**
DC	2	1	4	5	1	**13**
TOT.	**33**	**43**	**80**	**45**	**23**	**224**

some screenshots (taken during the process definition activity performed with the JaWE editor) are shown in Figure 4. After completing the process description, an administrator module is used for detailing the process definition. For each activity of the workflow it is necessary to specify: *(i)* the stakeholder that is responsible to work on it, and *(ii)* the external application used to support the production of the specified artefacts.

In order to complete the process design definition, each activity requires the definition of a set of rules: *(i)* auto-compositions are rules describing how to automatically build parts of the artefact that depend on the work executed in previous activities; *(ii)* semantic interpretation rules define how to extract semantic information from the artefact (for instance in this case study, if a package is found in a specific diagram, it has to be considered as an agent and use cases within it as requirements under the responsibility of that agent); *(iii)* semantic validation rules define semantic constraints to hold (as prescribed in the metamodel definition); *(iv)* syntactic validation rules define notational constraints to hold; finally *(v)* update rules are used to automatically compile some artefacts or maintain coherence among different models (for instance in case an element is renamed).

Table 1 summarizes the number of rules that we created for each category and for each activity of the PASSI design process. A total of 224 rules have been defined for the 13 activities. Semantic validation is the category that contains most of these rules. Just considering the number of rules per activity, the Agent Identification activity required the greatest number of rules, especially because several semantic validation rules are necessary for it. Transformation rules are not considered in Table 1 since all PASSI artefacts are

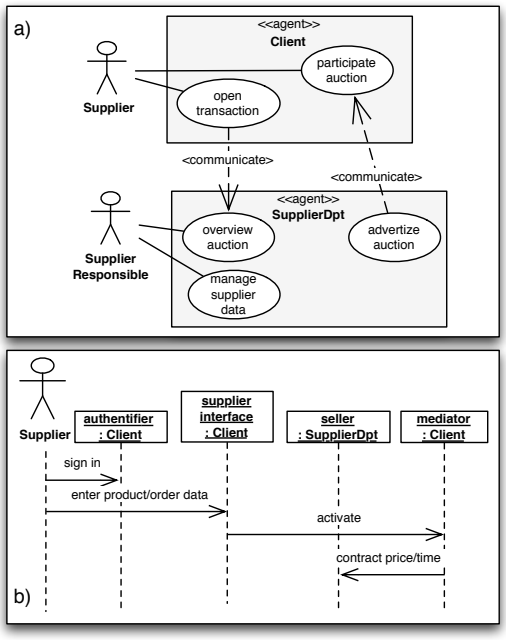

Figure 5: Two subsequent PASSI activities.

UML diagrams so these rules were reused from existing ones.

After this phase the administrator can *(i)* instantiate the PASSI process for one ore more projects, *(ii)* and monitor the execution of each instance. Designers access the list of activities assigned to them by using the UserAgent application; this latter in turn, receives the list of activities from the AdministratorAgent that extracts that from the process specification. More specifically, the list is synchronized with the design process state of progress, in according to the flow of activities defined in the previous phase as a workflow. If the generic activity A is not completed, all A antecedents are available for working, but all descendants are not in the list.

When a designer selects an activity from this list, the corresponding ActivityAgent tool is executed. This is responsible for correctly executing the activity: *(i)* interacting with external tools, *(ii)* providing the auto-composition of artefacts, *(iii)* verifying syntax and semantics of produced artefacts, and finally *(iv)* providing a semantic interpretation of artefacts.

Figure 5 reports two diagrams of a PASSI project concerning the design of a system supporting the supply chain of a manufacturing company.

The first diagram (shown in Figure 5.a) is an UML Use Case diagram representing the Agent Identification phase (in this diagram use cases are clustered in packages representing agents in terms of their functional responsibilities), whereas the second diagram (Figure 5.b) is a Sequence diagram related to the Role Identification phase. It is worth to consider that in PASSI: *(i)* the Roles Identification activity immediately follows the Agent Identification one, *(ii)* they are logically related, and *(iii)* they are based on conventional UML diagrams.

Figure 5.a shows a diagram where the subsystems repre-

sent two autonomous agents, and use cases represent responsibilities. Actors are traditional human stakeholders that communicate with the system, while communicate relationships between use cases indicate some dependencies (agents have to interact in order to execute functionalities). Figure 5.a shows two actors: the *Supplier* of the company and the *SupplierResponsible*, a clerk of the company responsible to manage relationships with external suppliers. Two agents are also shown, the *Client* and the *SupplierDpt*, that represent interfaces for their correspondent actors; the *Client* agent may have multiple instances, depending by the number of active suppliers: this agent is responsible to interact with the *Supplier*, and it allows him to participate to auction started from the manufacturing company. On the other hand, the *SupplierDpt* agent is responsible to manage active auctions and to inform participants of auction results.

Table 2: Results of the semantic interpretation.

	Element	MM Element
Agent Identification	Supplier	Actor
	SupplierResponsible	Actor
	SupplierAgent	Agent
	SupplierDptAgent	Agent
	Open transaction	Responsibility
	Partecipate auction	Responsibility
	Overview auction	Responsibility
	Advertize auction	Responsibility
	Manage supplier data	Responsibility
Role Identific.	Authentifier	Role
	Supplier interface	Role
	Seller	Role
	Mediator	Role

When each of the activities is completed, the ActivityAgent executes the syntax and semantic validations that in our example we will suppose do not fire error/warning messages. Semantic interpretation gives the results shown in Table 2. Notational elements from Figure 5.a are correctly interpreted as instances of the system metamodel elements and then stored in the knowledge base.

As already said, Roles Identification is the subsequent activity prescribed by the PASSI design process and one example of resulting diagram is represented in Figure 5.b. The sequence diagram illustrates a possible scenario for the system-to-be. Actors are users of the system, whereas participating objects represent roles that agents may play in the scenario. Each object is identified by an instance name that represents the role, whereas the class name is a reference to the agent. The sequence diagram in Figure 5.b illustrates the interactions occurring when a *Supplier* wish to put up raw materials for a supplier auction. Three roles have been introduced for the *Client* agent (*authentifier*, *supplier_interface* and *mediator*) whereas the *SupplierDpt* agent only plays the *seller* role.

When this activity is completed too, the ActivityAgent executes the semantic interpretation again, and the model is enriched with new elements shown in Table 2.

Some concluding remarks: PTK (PASSI toolkit) was developed in six person-months (PMs hence after) and its completion required about another person-year for a total of 18 PMs of effort. By using Metameth the development of the PASSI workflow and the definition of the Protege' ontology required 1 PM, whereas the definition of rules (summarized in Table 1) required other 2 PMs, for a total of 3 PMs for completing the customisation of the tool for supporting PASSI. In addition, each major change in the PTK tool implied an average effort of 3 PMs, whereas the introduction of a new activity in Metameth usually requires one person-week only.

4. RELATED WORKS

Several tools operating in the same field of action of Metameth today exist. Each of them provides support to the method engineer's work in different ways and presents different features; for space reason we limit our analysis to the most complete we know: Metaedit+ [15], it gives the possibility of implementing a domain specific modelling language and using it in a ad-hoc created CASE tool. Metaedit+ is at the same time a CAME tool and a CASE tool, it is the only tool we know that allows the instantiation of a CASE tool starting from the definition of a precise modelling language but it does not offer support for managing a design process.

Metaedit+ is based upon a meta-modelling language, GOPRR and it can supply documentation and code generation, besides it allows multiple users and presents the principle advantage of supplying an easy way for creating and using each kind on notation but there is no intelligent system support, no consistency checks on documents, and no semantic and syntactic checks on the produced artefact.

Other existing tools allow to manage the method fragments definition and assembly and to instantiate a CASE tool, they provide some kind of checks on the produced artefact but none of them is assisted by an intelligent system for supporting the designer.

The Metameth tool tries to collect the advantages of the known approaches in the fact that it allows the definition of a domain modelling specific language (in form of an ontological metamodel) and the instantiation of a CASE tool that is specific for each process. Moreover it exhibits some other features:
(i) the tool is aware of the prescribed design process and facilitates in following the prescribed flow of design activities;
(ii) the flow of information exchanged among the different components of the tool (mainly the model graphic editor and the expert system) is fluently enabled by a set of format transformations that are transparent to the designer as well as the expert system;
(iii) the presence of an expert system allows sophisticated checks on design models consistency, completeness and respect of semantic rules defined in the metamodel;
(iv) the automatic composition of some artefacts (again enabled by the expert system) by exporting in the proper way portions of knowledge from the expert system allows the designer to save time, to focus on essential tasks and increases the quality of the results.
(v) The tool allows the participation of several designers working on the same project at the same time and an easy condivision of produced artefacts among them.
(vi) the extension of the tool to support different design model editors (and therefore different modelling notations) as well as other tools (text editors and other specific analysis tools) is possible and only conditioned by the coordination of the domain specific ontological metamodel with the modelling notation (this is realized by means of the semantics understanding rules).

5. CONCLUSIONS

In this paper we presented the tool (Metameth) we developed for supporting the development and the enacting of a design process following the paradigm of Situational Method Engineering. Situational Method Engineering (SME) provides means for creating ad-hoc design processes by assembling pieces of process (the fragments) coming from existing ones.

SME is supported by CAME tools allowing the definition, retrieval from repository and assembly of fragments; besides when a new design process is created a specific CASE tool has to be instantiated from the CAME tool in order to give the right support for each created process.

This is the key point of the work we propose in this paper: a tool able to support the method engineer and that can be instantiated as a specific CASE tool also allowing intelligent support to the designer.

In addition to the CAME features we provided our tool with the possibility (CAPE tool characteristics) of managing the whole process by orchestrating the activities it is composed of. The tool also enables the collaboration of different stakeholders in these design activities and helps in reducing the time the method engineer spends for producing the specific CASE tool.

In the future we will introduce more features above all in the process definition part where we are planning to provide some kind of support for the design rules modeling and we will integrate the CAME we developed for interfacing the method engineer with the repository we constructed.

6. ACKNOWLEDGMENTS

Part of this work makes use of results produced by the PI2S2 Project managed by the Consorzio COMETA, a project co-funded by the Italian Ministry of University and Research (MIUR) within the Piano Operativo Nazionale "Ricerca Scientifica, Sviluppo Tecnologico, Alta Formazione" (PON 2000-2006).

7. ADDITIONAL AUTHORS

Additional authors: S. Gaglio (Dipartimento di Ingegneria Informatica, Universitá degli Studi di Palermo, email: `gaglio@dinfo.unipa.it`).

8. REFERENCES

[1] The workflow management coalition. http://www.wfmc.org/.

[2] S. Brinkkemper, M. Saeki, and F. Harmsen. A Method Engineering Language for the Description of Systems Development Methods. *Proceedings of the 13th International Conference on Advanced Information Systems Engineering*, pages 473–476, 2001.

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

[4] M. Cossentino. From requirements to code with the PASSI methodology. In *Agent Oriented Methodologies*, chapter IV, pages 79–106. Idea Group Publishing, Hershey, PA, USA, June 2005.

[5] 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.

[6] M. Cossentino, S. Galland, S. Gaglio, N. Gaud, H. Hilaire, A. Koukam, and V. Seidita. A mas metamodel-driven approach to process composition. In *Proc. of the Ninth International Workshop on Agen-Oriented Software Engineering (AOSE-2008)*, 2008.

[7] M. Cossentino, L. Sabatucci, and A. Chella. Designing jade systems with the support of case tools and patterns. *Special Issue on JADE of Telecom Italia Journal EXP of September.*, 2003.

[8] M. Cossentino, L. Sabatucci, V. Seidita, and S. Gaglio. An expert system for the design of agents. In *Proceedings of International Workshop on Agent Supported Cooperative Work at the IEEE The Second International Conference on Digital Information Management (ICDIM'07)*, 2007.

[9] 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.

[10] A. Harmsen, M. Ernst, and U. Twente. *Situational Method Engineering*. Moret Ernst & Young Management Consultants, 1997.

[11] M. Saeki, K. Iguchi, K. Wen-yin, and M. Shinohara. A Meta-Model for Representing Software Specification & Design Methods. *Proceedings of the IFIP WG8. 1 Working Conference on Information System Development Process*, pages 149–166, 1993.

[12] 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.

[13] S. Si-Said, C. Roland, and G. Grosz. Mentor: A computer aided requirements engineering environment. In *In Proceedings of the 8th international Conference on Advances information System Engineering*, volume 1080, pages 22–43. Lecture Notes In Computer Science. Springer-Verlag, May 20 - 24 1996.

[14] J.-P. Tolvanen. Incremental method engineering with modeling tools: Theoretical principles and empirical evidence (ph.d. thesis). *Jyvskyl Studies in Computer Science*, page 301, 1998.

[15] 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.