A Methodological Framework and Software Infrastructure for the Construction of Software Production Methods[1]

Mario Cervera, Manoli Albert, Victoria Torres, Vicente Pelechano

Centro de Investigación ProS,
46022 Valencia, Spain
{mcervera,malbert,vtorres,pele}@pros.upv.es
http://www.pros.upv.es

Abstract. The theory of Method Engineering becomes increasingly solid, but very few engineering tools have been developed to support the application of its research results. To overcome this limitation, this paper presents a methodological framework based on Model Driven Engineering techniques. The framework provides a method supported by a software platform for the construction of software production methods. This framework covers from the specification of the software production method to the generation of the CASE tool that supports it. This generation process has been semi-automated through model transformations. The CASE tool and the software platform are based on the Eclipse-based MOSKitt tool. The plugin-based architecture and the integrated modelling tools included in the MOSKitt tool turn it into a suitable software platform to support our proposal. To validate the proposal we have applied the framework to a case study.

Keywords: Method Engineering, Model Driven Engineering, CAME Tool, Eclipse Platform

1. Introduction

Method Engineering (ME) is defined as *the engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems* [1]. A lot of research work has been developed in this area during the last two decades, especially in the adaptation of Software Production Methods (here after SPM) to a specific situation, area that is known as Situational Method Engineering (SME). Most of the approaches developed in these two areas propose the construction of SPMs by assembling reusable fragments (parts that compose a method). Regarding the fragments assembly we find works focused on the description of these fragments (as it is the case of the MEL language proposed by Brinkkemper in [1]) and on the study of techniques for selecting ([3], [7]) and assembling ([2], [14], [15]) efficiently the fragments stored in a repository. Nevertheless, these approaches are difficult to put into practice, mainly because they are neither supported by software tools nor complete solutions for the design and implementation of SPMs (generation of CASE tool support for the designed SPM). This software support is usually provided in the shape of CAME environments (CAME stands for Computer-Aided Method Engineering and refers to the tools supporting ME approaches) which combine theoretical proposals with tool support for the construction of SPMs. However, to our knowledge, the existing environments constitute incomplete prototypes that only cover some parts of the ME process. This fact hinders the expected industrial adoption and success considering the benefits that this type of tools would provide to industry.

In this paper, we provide a methodological framework that allows designing and implementing SPMs. This is possible since the framework includes a set of engineering tools that assist during the development of the SPM. To build this framework we apply the main ideas of the most relevant approaches developed in ME (SPM construction through fragment assembly, usage of a reusable fragments repository, process enactment, etc.). The proposed framework has been designed following the Model Driven Engineering (MDE) principles. These principles advocate for the intensive use of models and model transformations to perform system development. So, the SPM design is done by specifying the model of the SPM which describes the process of the SPM. The key concepts of this model are *tasks* (work done during the process) and *products* (input and/or output of tasks). The construction of this model has been defined in two steps. During the first step, the method engineer creates a generic SPM model that does not include details such as how tasks have to be performed. Then, in the second step, the method engineer instantiates the generic model by providing information about the languages, notations

[1] This work has been developed with the support of MEC under the project SESAMO TIN2007-62894 and cofinanced by FEDER.

or techniques used in each task or product. This approach allows the method engineer to make different configurations of the same generic description of the SPM according to special features of the different projects or development teams. Once the SPM model is completed, model transformations are used to automate the generation of the CASE tool supporting the SPM. The generated CASE tool includes: (1) all the software components that provide support to the tasks and products of the SPM and (2) the software component that provides support to the process of the SPM. The last component allows the software engineer to keep the state of a SPM instance and to guide him/her during its execution. These two aspects of the CASE tool correspond to the two parts in which a SPM is divided (the *product* part and the *process* part).

The process followed to build the proposed framework has applied a bottom-up strategy. It implies that the process begins by identifying the characteristics of the CASE tools that our framework should obtain. In order to determine these characteristics we have developed a CASE tool that supports a specific SPM. Once these characteristics have been identified, they have been used to determine the concepts required to specify SPM and further on to define the framework. The CASE tool has been built using the MOSKitt tool [12], an Eclipse-based tool which includes (1) a plugin-based architecture and (2) a set of integrated modeling tools (e.g. EMF [5]). The characteristics of the MOSKitt tool turn it into a suitable software platform to support our proposal, i.e. to build on it the CAME and CASE tools.

The main contribution of this work is to provide a methodological framework to help and assist method engineers not only in the definition of SPM but also in deriving CASE tools to support them. Among the characteristics of the framework we highlight (1) the intensive use of standards, (2) its integrated design being all the process carried out with the same technology (the MOSKitt tool), (3) the automation degree achieved thanks to the application of MDE techniques, (4) the reusability degree of the SPM models (specifying generic SPM that can be configured to suit it to a specific context of use), and (5) the generation of CASE tools that support process enactment.

The remainder of the paper is structured as follows. Section 2 briefly presents the state of the art of the ME and SME making emphasis in the limitations found in the existing proposals. Then, section 3 introduces an overview of our proposal. Section 4 presents in detail the methodological framework designed to cover the limitations found in ME/SME areas. Section 5 presents a case study where we apply the proposed methodological framework. Finally, section 6 provides some conclusions and outlines further work.

2. State of the art

Kumar and Welke developed the first research work in the area of the ME during the early nineties [10]. Since then, many authors have contributed to this area, in particular to the most theoretical part of the engineering. Among these contributions we find (1) new concepts and terms that were also applied to the SME [1], [9], (2) different techniques to specify methods, for instance by assembling existing fragments [11], [14], [15] or (3) different techniques to specify methods for specific projects [8]. All these works constitute a good proof of the amount of theoretical work developed in this area. However, the existing proposals have not been successfully exploited in industry, being relegated just to educational environments. The main cause of this reality is the unbalanced effort made between the research devoted just to the specification of SPM and the research devoted to the construction of software tools for the support of SPM (this is, for the construction of CASE tools). Surveys such as the one presented in [13] put in evidence this reality, and demonstrate that most of the support is provided as prototype tools that just cover part of the corresponding SPMs.

Fig. 1 presents the components of a CAME environment[2]. Checking the literature, we noticed that none of the existing proposals (1) is accompanied with a CAME environment that provide these components (2) or provides an automated process to generate the CASE tool and the Process Support Environment. This can be due to the fact that all these proposals have been designed following a top-down approach, where the development process begins defining the concepts for specifying SPM and then the generation of tool support is tackled. However, this is a very difficult task, and actually none of the proposals has been successfully developed. Therefore, to improve this situation we have proceeded

[2] In this figure, the CAME environment has been divided in two parts, the CAME part, which provides facilities for ME, and the CASE part, which offers means for the generation of CASE tools and process support environment.

the other way round, this is, following a bottom-up strategy. We have developed a CASE tool for a specific case study SPM and have identified the characteristics of this tool. These characteristics have been abstracted to determine the concepts required to specify a SPM and further on to build the framework. The CASE tool has been developed using the Eclipse-based tool MOSKitt which has demonstrated us the feasibility of the Eclipse platform, and the plugin architecture in which it is based on, for the support of our ME proposal.

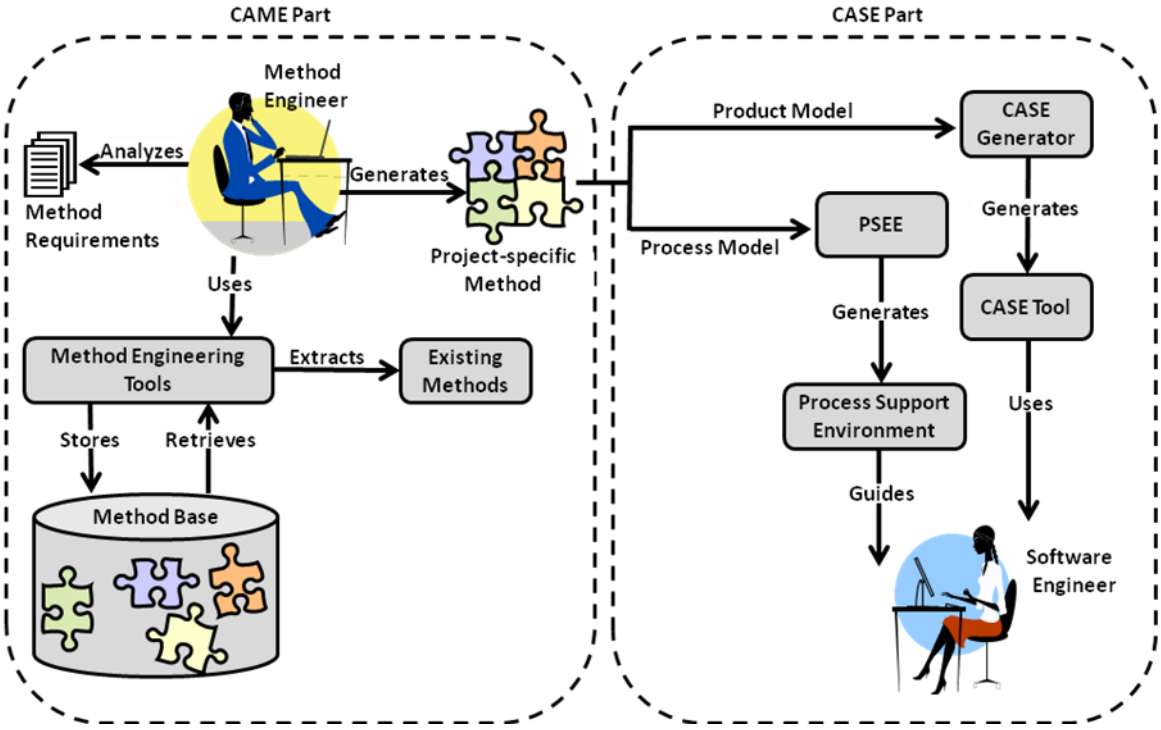

Fig. 1. General architecture of CAME environments from [13]

Trying to solve the limitations found in ME context, this work proposes a methodological framework that, applying the techniques developed in the MDE area, helps and assists the method engineer in both, the specification of SPM and the construction of CASE tools for the support of such methods.

3. Overview of the proposal

The developed framework provides a methodology together with a software infrastructure (CAME environment) aimed at supporting the design and implementation of SPMs. The methodological framework has been organized in three parts, which are *method design*, *method configuration* and *method implementation* (see Fig. 2). These parts constitute the phases that have to be sequentially performed to build a SPM:

- *Method design*: The method engineer builds the *Method Model* by identifying all the elements (tasks, products, etc.) involved in the process of the SPM. These elements can be defined from scratch or selected from a *method base* repository, which contains reusable elements that were specified in other SPM. This first version of the model constitutes a *generic description* where no specific languages or notations are specified for the elements of the model.

- *Method configuration*: The method engineer associates the elements of the *Method Model* built in the previous phase with assets stored in an *asset base* repository. This repository contains models, metamodels, transformations, etc., which have been built either in other SPM or ad-hoc for the SPM under construction (the method engineer can use the tools provided in our CAME environment for this purpose). These assets turn the generic elements into specific ones where languages and notations are specified. The partition of the SPM specification in two phases (*method design* and

method configuration) allows the method engineer to take generic descriptions and perform different configurations according to each particular target project or development team.

- *Method implementation*: During this phase a set of model transformations is executed to automatically obtain either the complete CASE tool or part of it when the generation cannot be fully automated. In the latter situation, the method engineer can take part in the process and complete the tool by using the tools provided in our CAME environment for this purpose.

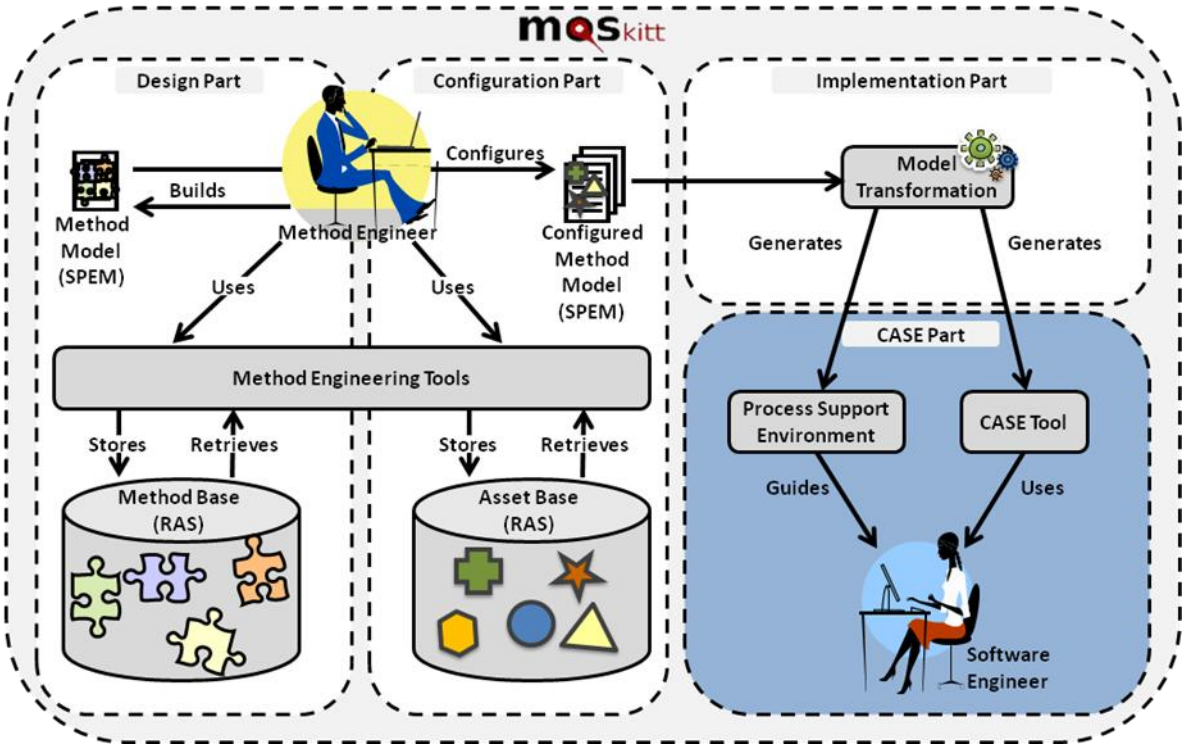

Fig. 2. Methodological Framework

To perform these three phases, we use as much as possible the available standards. To specify SPM we use SPEM 2.0 [17], which is the standard developed by the OMG to define software process methods, and to define both repositories we use RAS [16], the OMG specification that provides a standard way to assembly, organize, store and document reusable software assets.

We provide a software infrastructure, which is a set of method engineering tools, to support the three phases of the framework. This infrastructure assists the method engineer during the method design and method configuration phases, and automates a large part of the method implementation phase. The tools provided in the software infrastructure are editors, repositories and guides, which are detailed in the next section.

4. A Methodological Framework for the Construction of Software Production Methods

This section presents in detail the three phases that compose the methodological framework. Each phase is described first by introducing its goal and the procedure to achieve it and then by introducing the software infrastructure supporting each phase.

4.1. Method Design Phase

This phase corresponds to the *design part* of Fig. 2. During this phase, the method engineer builds the *method model*, which can either be built from scratch or by reusing method fragments from the *method base* repository. This model describes the process of the SPM by using concepts such as:

- **Task**: It represents an activity performed during the execution of a SPM instance (e.g. *business process analysis, web specification,* etc.).
- **Product**: It represents an artefact that is either consumed or generated in a task (e.g. *business process model, structural model,* etc.).
- **Role**: It represents an agent that participates in a SPM performing different tasks. This can refer to a human being agent (e.g. *analyst, developer,* etc.) or to an automated system.
- **Flow Connector**: It represents the order in which two associated tasks (each one in a different end) are executed.
- **Gateways**: It represents points within the SPM where the flow is diverged or converged depending on the gateway type.
- **Guide**: It is a document that provides some assistance to perform a task or to manipulate a specific product.

All these concepts specify *who* (Role), *when* (Flow Connector and Gateways) and *how* (Task, Product and Guide) the process of the SPM is performed. In this work, to specify the *method model* according to these concepts we have used SPEM 2.0, the standard developed by the OMG for the definition of SPM.

As a result of this phase we obtain the *method model* which is a SPM description that does not include specific techniques or languages/notations associated to its elements (e.g. we can specify the product *business process model* which does not refer to any technique or notation). These details will be provided in the next phase (*method configuration*). The main benefits of this separation are that: (1) we keep generic definitions of SPM (which means that we can take this generic definition and perform different method configurations according to each particular target project or team) and (2) it stresses the importance of reusability.

4.1.1. Software Infrastructure

We provide support to the Method Design phase by means of the following tools:

- *An editor to build the method model*: We use the EPF Composer editor (EPFC), a SPEM 2.0 editor provided in the EPF project [6], which has been integrated in MOSKitt. By using this editor we can build SPM models according to the SPEM metamodel. These models describe the process of the SPM, which can be described associating UML Activity Diagrams to the own model, and the products involved in the process.

- *A method base repository*: Method fragments are stored in a *method base* repository following the RAS standard (which implies that the fragments have to be packaged using this specification). To reuse the fragments of this repository in the *method model,* it is necessary to extend the EPFC editor. This extension allows the user to retrieve fragments from the repository during the specification of the *method model*. To retrieve the fragments in an efficient way, we make use of some of the ideas proposed in [15]. According to these ideas, our method fragments are characterized by (1) a *descriptor,* which provides information about origin, objective and type of the fragment and, (2) an *interface* which is a couple <situation, intention> that characterises the situation that is the input of the fragment and the intention (goal) that the fragment achieves.

- *A guide to build the method model*: A wizard is provided to guide the method engineer through the performance of the *method design* phase.

4.2. Method Configuration Phase

Once the *method design* phase has been finished, the method engineer should add to the *method model* some information related to its execution (*configuration part* of Fig. 2). This is necessary since no information regarding the technique or language/notation was given during the previous phase. Therefore, some of the products and tasks included in the *method model* have to be associated with assets that support them. These assets are contained in the *asset base repository,* which stores metamodels, transformations, editors, etc. With this association, the method engineer instantiates the generic model

into a specific one. For example, a *business process model* product identified during the *method design* phase refers to a generic model, which could be built in any language or notation. Later on, during the *method configuration* phase, this product is associated to the BPMN metamodel (which is an asset stored in the *asset base* repository). Thus the method engineer instantiates the generic *business process model* into a concrete notation, which is in this case the BPMN notation.

It is possible that the asset required by the method engineer is not stored in the *asset base repository*. In this case, the flexibility of the plugin-based architecture in which Eclipse relies on allows the method engineer to build this asset. Then, this asset is stored in the repository so that the method engineer can use it in the configuration of the *method model*.

4.2.1. Software Infrastructure

This phase is supported by the following tools:

- *An asset base repository*: To build the assets contained in this repository, we use again the ideas proposed in [15]. To link these assets with the elements of the *method model* (tasks and products), we have extended the SPEM class *ReusableAsset* so its instances can reference the assets from the repository.

- *A guide to configure the method model*: A wizard is provided to guide the method engineer through the performance of the *method model* configuration.

4.3. Method Implementation Phase

The *method implementation* phase corresponds to the *implementation part* of Fig. 2. During this phase the *configured method model* is used to systematically generate the CASE tool that supports the method. This support is twofold:

- A *CASE tool* that contains all the necessary components to support the product part of the SPM.

- A *process support environment* that provides support to the process of the SPM. This support turn the *CASE tool* into a real project environment where users can launch new SPM instances or execute existing ones. The environment allows launching guides that assist during the execution of a specific task, checking the dependences between products or tasks, etc.

4.3.1. Software Infrastructure

This phase is supported by the following tool:

- A *model-to-text (M2T) transformation*: a M2T transformation integrated in MOSKitt. This transformation takes as input the *configured method model* built in the previous phases and obtains an Eclipse *product configuration file*. This file contains all the necessary information to build a MOSKitt reconfiguration containing all the support for the SPM (*the CASE tool* and *process support environment*). The transformation has been implemented using the XPand language [19].

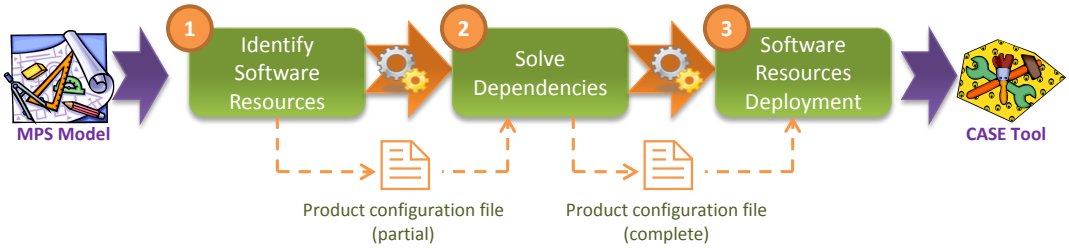

Fig. 3. Case Tool Generation Process

Fig. 3 depicts the steps that the transformation automatically performs to obtain from the *configured method model* its corresponding CASE tool support. The details of the steps are the following:

Identify Software Resources: Identifying the software resources that have to be deployed upon the MOSKitt platform to support the products and tasks involved in the SPM. This set of resources is included in a *product configuration file*, which is the result of this step.

Solve Dependencies: Solving the potential conflicts that can arise due to software dependencies. Therefore, all the software resources required by the resources identified in the previous step are also included in the product configuration file.

Software Resources Deployment: Generating the CASE tool from the product configuration file by using the Eclipse Plugin Development Environment (PDE).This tool is a MOSKitt reconfiguration that just includes the plugins strictly necessary to provide support to the SPM definition.

The use of MDE techniques and technologies enables the automation of part of the Method Implementation phase. As SPMs are described in terms of models according to a particular metamodel (in our case SPEM 2.0), model-to-text transformations can be applied to produce the *configuration file* that allows us to reconfigure MOSKitt to support a specific SPM.

5. The Methodological Framework in Practice

In this section the methodological framework is applied to a SPM that is introduced in [18], which proposes the steps for the construction of web applications to support a set of business processes. We present the *method model* specification and its implementation using the CAME environment supporting our methodological framework.

5.1. Method Model specification

The specification of the SPM of the case study is done by building the *method model*. In a first step the *method model* is specified without detailing techniques, languages or notations (*method design* phase). So, we define the tasks (*Business process analysis, system specification,* etc.), products (*Business process model, services model*, etc.) and roles (*Analyst, Developer,* etc.), together with the workflow (process part) of the SPM. To do so, we use the tools provided in our approach to build the *method model* (the EPFC editor, the *method base* repository and the guide to perform the method design).

The *method model* specification of the case study is shown in Fig. 4[3] (which contains a screenshot of the EPFC editor integrated in MOSKitt). In this figure the "Library" view is displayed on the left. This view contains all the elements of the SPM in a tree viewer. Specifically, this view shows all the different roles, tasks and the I/O resources consumed and produced by the different tasks. These I/O resources correspond to the product part of the SPM. On the right side of Fig. 4 the graphical representation of the SPM is shown as an UML activity diagram, where roles appear as *Activity Partition* elements and tasks are distributed along the SPM workflow. This workflow corresponds to the process part of the SPM. Furthermore, aside from what is shown in the figure, the EPFC editor provides other kind of views and editors (graphical and form-based) for the creation and deletion of SPM elements and the edition of its properties.

[3] Available also at www.dsic.upv.es/~vtorres/icsp2010.

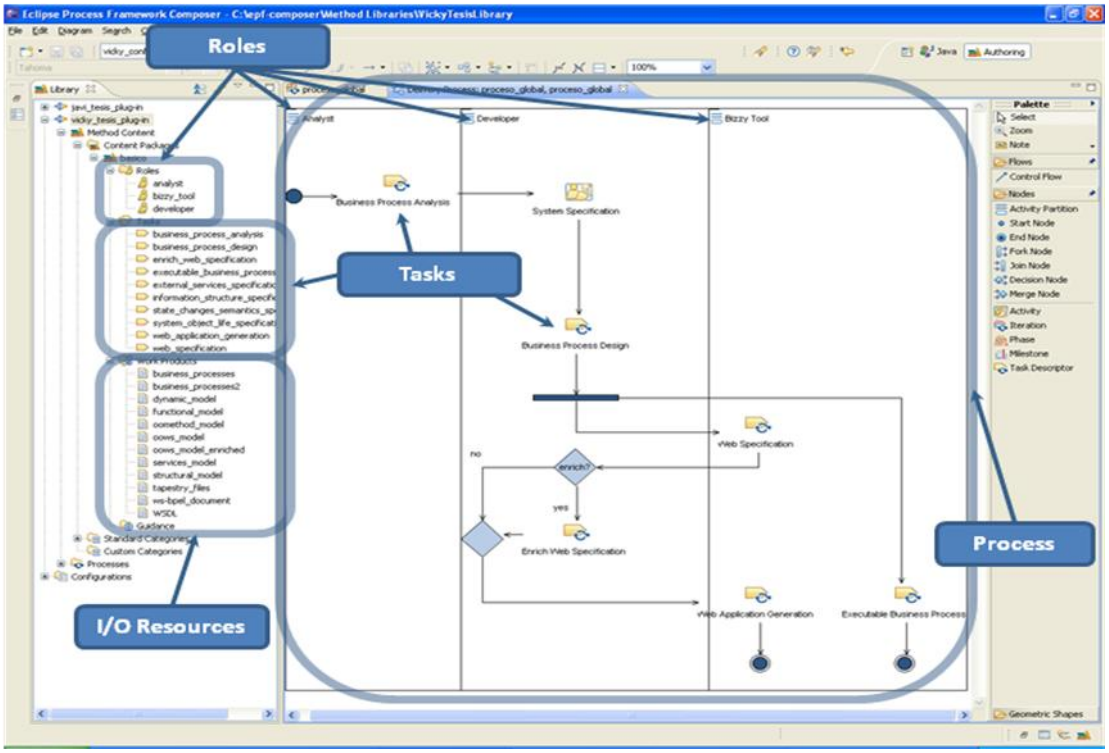

Fig. 4. Case study specification with EPF Composer

Once the *method model* has been built (i.e., the *method design* phase is finished), we configure it (*method configuration* phase). To do so, we define the links between the elements of the model (tasks and products) and the assets that support them, which are stored in the *asset base repository*. Table 1 and Table 2 show the links between the tasks and products specified in the *method model* of the case study and the assets of the *asset base repository*. The symbol * points out that the asset was initially available in the repository, whereas the symbol ** points out that the asset is defined by the method engineer.

Task	Asset
Business Process Analysis	Guide with the steps of the task *
System Specification	Guide with the steps of the task *
Business Process Design	Guide with the steps of the task **
Web specification	M2M Transformation **
Enrich Web Specification	Guide with the steps of the task **
Web Application Generation	M2T Transformation **
Executable Business Process	M2M Transformation **

Table 1. Tasks and Assets of the SPM

Product	Asset
Business Process Model	BPMN Metamodel *
Conceptual Model	OO-Method Metamodel *
Navigational and Presentation Models	OOWS Metamodel *
Tapestry Code	Text Editor *
WS-BPEL Document	Text Editor *
Services Model	Services Metamodel **

Table 2. Products and Assets of the SPM

The links between SPM elements and the assets of the *asset base* repository are established through instances of the SPEM class *ReusableAsset*. An example of this link is shown in Fig. 5.

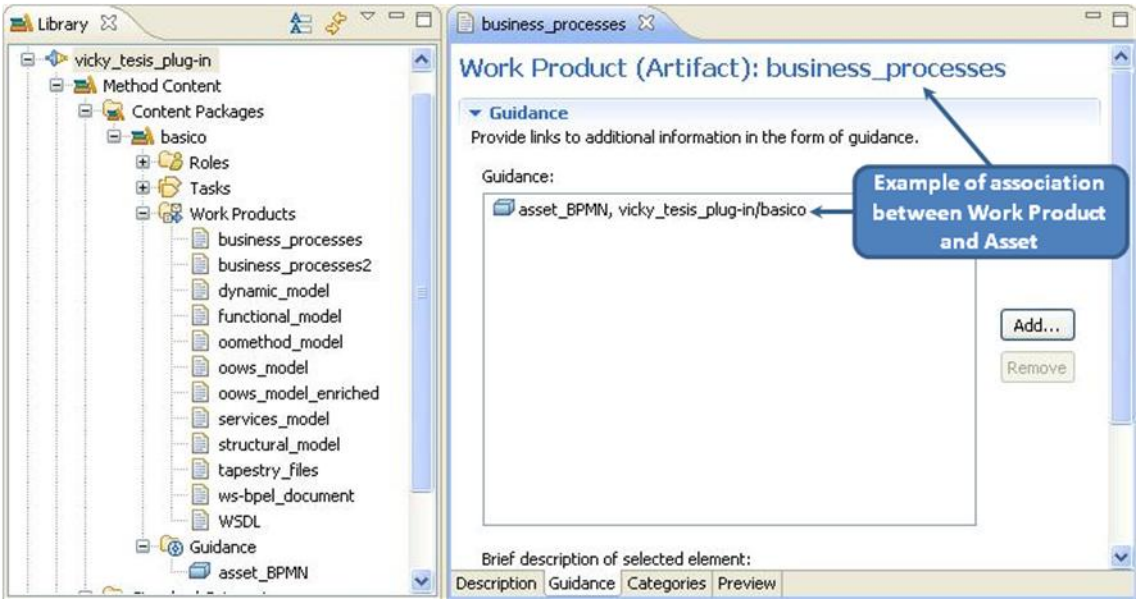

Fig. 5. Association between a product and an asset

5.2. Generated Case Tool support

An Eclipse application (Rich Client Application) is obtained from the *method model* specification. This application represents the CASE tool support (*the CASE tool* and *process support environment*) for the SPM. The CASE tool is a MOSKitt reconfiguration that includes all the plugins supporting the product and process parts of the SPM. A screenshot of this tool is shown in Fig. 6. Several views can be observed in the figure. These views are the following:

- **Product Explorer:** shows in a tree viewer the products consumed/produced/modified by the ongoing tasks and the tasks that have already been executed. This view can be filtered by role so the user only see the products he/she is responsible for. By selecting the product element, the associated software resource is opened to edit the product.

- **Process:** shows in a tree viewer the different tasks that can be executed in the current moment. This view can be filtered in a similar way as the Product Explorer so the user only see the tasks he/she is responsible for. When a task is selected in this view:

    - If this task generates a product, then this product is created and added to the "Product explorer".
    - If this task consumes a product and this has not been generatead previously in the process, the user has to specify its file system location. On the contrary, if the product has already been created in the process, the user has to indicate which product from the available ones (those appearing in the Product Explorer view) is consumed in the task.

- **Guides:** shows the list of the guides associated to the product or task selected in the corresponding view. The main goal of these guides is to assist the user during the manipulation of the product or the execution of the associated task. There may be more than one guide associated to a product/task as shown in Fig. 6 and even different types of guides (i.e. Eclipse cheat sheets, text files, hyper-text files, etc.). When a guide is selected, it is opened with the corresponding editor.

- **Product Dependencies:** shows the list of dependencies of the product or task selected in the corresponding view. These dependencies are grouped by role allowing the user to know the existing dependencies between the products manipulated by him and also by other roles.

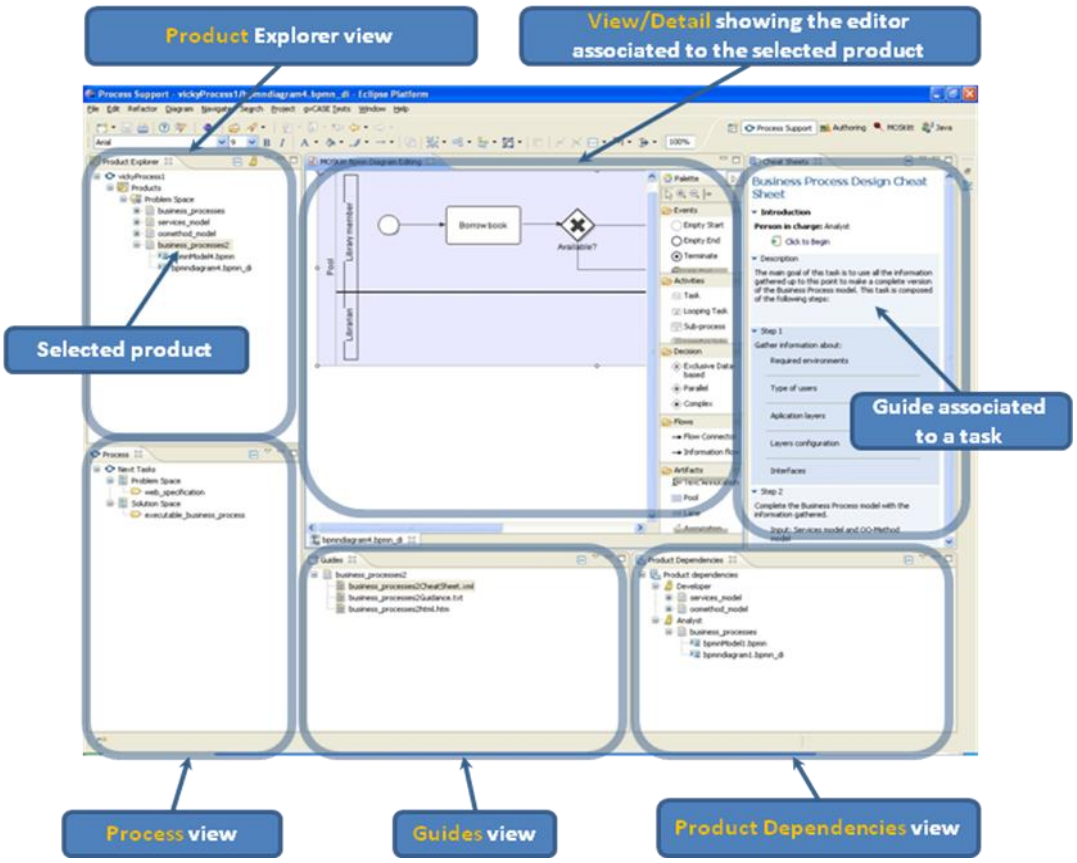

Fig. 6. Final CASE tool

6. Conclusions and further work

In this paper a methodological framework for the construction of SPM covering from its specification to its implementation (generation of the CASE tool support) has been presented. The framework includes tool support that has been built on the MOSKitt platform. The combination of ME and MDE techniques together with the flexibility of the plugin-based architecture of MOSKitt allows us to obtain a framework that enables the complete construction of SPM (obtaining a CASE tool which provides complete support to the specified SPM). The specification of SPM focuses on its process part. To build the method model, the SPEM [17] and RAS [16] standards have been used. The method model is built in two phases, the method design where the method is described avoiding the specification of techniques or languages used in the process, and the method configuration, where the method design is completed with assets that determine specific techniques or languages for the elements of the method model. From the method model, a model transformation semi-automatically obtains the CASE tool that supports the SPM. This CASE tool is a MOSKitt reconfiguration that includes the necessary plugins to support the SPM. An important issue of this CASE tool is the process support included in it, which guides the software engineer during a project execution.

Regarding further work, we are working on the improvement of the set of tools that supports the methodological framework. In addition, we are concerning with one of the big challenges of ME [20], which takes into account the variability of the SPM both at modeling level and runtime. Variability appears as a relevant challenge in ME, since it is very common that context changes entailing SPM adaptation during the progress of a project. So, we are working on providing mechanisms to support this variability. At modeling level we propose the use of techniques based on fragment substitution to specify this variability. These techniques allow us to keep separately the common and variable parts of the SPM, which makes the models more legible and easier to specify and maintain. At implementation level, we propose the introduction in MOSKitt of a reconfiguration engine (for instance, MoRE [4]) to allow the CASE tool reconfiguration at runtime based on context changes.

7. References

[1] Brinkkemper, S. Method Engineering: Engineering of Information Systems Development Methods and Tools. *Information and Software Technology, Elsevier, 38*, 275-280 (1996).

[2] Brinkkemper, S.; Saeki, M. & Harmsen, F. Meta-Modelling Based Assembly Techniques for Situational Method Engineering. *Inf. Syst., 24*, 209-228 (1999).

[3] Brinkkemper, S.; Saeki, M. & Harmsen, F. A Method Engineering Language for the Description of Systems Development Methods. *CAiSE '01: Proceedings of the 13th International Conference on Advanced Information Systems Engineering, Springer-Verlag*, 473-476 (2001).

[4] Cetina, C.; Giner, P.; Fons, J. & Pelechano, V. Autonomic Computing through Reuse of Variability Models at Runtime: The Case of Smart Homes. *Computer, IEEE Computer Society Press, 42*, 37-43 (2009).

[5] Eclipse Modeling Framework Project. http://www.eclipse.org/modeling/emf/

[6] Eclipse Process Framework Project (EPF), http://www.eclipse.org/epf/

[7] Harmsen, F. & Brinkkemper, S. Design and Implementation of a Method Base Management System for a Situational CASE Environment. *Asia-Pacific Software Engineering Conference, IEEE Computer Society, 0*, 430 (1995).

[8] Henderson-Sellers, B. Method engineering for OO systems development. *Commun. ACM, ACM, 46*, 73-78 (2003).

[9] Ter Hofstede, A. H. M. & Verhoef, T. F. On the feasibility of situational method engineering. *Inf. Syst., Elsevier Science Ltd., 22*, 401-422 (1997).

[10] Kumar, K. & Welke, R. J. Methodology Engineering: A Proposal for Situation-Specific Methodology Construction. *Challenges and Strategies for Research in Systems Development, John Wiley & Sons, Inc.*, 257-269 (1992).

[11] Mirbel, I. & Ralyté, J. Situational method engineering: combining assembly-based and roadmap-driven approaches. *Requir. Eng., Springer-Verlag New York, Inc., 11*, 58-78 (2005).

[12] MOdeling Software Kitt (MOSKitt), http://www.moskitt.org

[13] Niknafs, A. & Ramsin, R. Computer-Aided Method Engineering: An Analysis of Existing Environments. *CAiSE,* 2008, 525-540.

[14] Ralyté, J. & Rolland, C. An Assembly Process Model for Method Engineering. *CAiSE '01: Proceedings of the 13th International Conference on Advanced Information Systems Engineering, Springer-Verlag*, 267-283 (2001).

[15] Ralyté, J. & Rolland, C. An Approach for Method Reengineering. *ER '01: Proceedings of the 20th International Conference on Conceptual Modeling, Springer-Verlag*, 471-484 (2001).

[16] Reusable Asset Specification (RAS) OMG Available Specification version 2.2. OMG Document Number: formal/2005-11-02.

[17] Software Process Engineering Meta-model (SPEM) OMG Available Specification version 2.0. OMG Document Number: formal/2008-04-01.

[18] Torres, V. A Ph.D. thesis entitled "A Web Engineering Approach for the Development of Business Process-Driven Web applications". Technical University of Valencia, 2008. Available at http://www.dsic.upv.es/mapa/ingles/desctesis.pl?tesis=etd-04112008-140714

[19] Xpand, http://www.eclipse.org/modeling/m2t/?project=xpand

[20] Armbrust, O.; Katahira, M.; Miyamoto, Y.; Münch, J.; Nakao, H. & Ocampo, A. Scoping Software Process Models - Initial Concepts and Experience from Defining Space Standards. *ICSP,* 2008, 160-172.