The MOSKitt4ME Approach: Providing Process Support in a Method Engineering Context*

Mario Cervera, Manoli Albert, Victoria Torres, and Vicente Pelechano

Centro de Investigación en Métodos de Producción de Software
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain
{mcervera,malbert,vtorres,pele}@pros.upv.es

Abstract. It is commonly agreed that software developments methods must be defined (or adapted) in-house in order to meet the particular needs of the organizations where they are to be applied. To help meet this challenge, Method Engineering (ME) research aims to provide solutions to efficiently deal with the definition and adaptation of methods, and the construction of the supporting software tools. However, while the product part of methods is fully considered by most ME approaches, the specification and enactment of the process part is less well-supported. To fill this gap, this work presents a methodological ME approach and a Computer-Aided Method Engineering (CAME) environment (MOSKitt4ME) that support the design and implementation of the process part of methods in the context of Model-Driven Engineering. The proposal is illustrated by means of a real case study that is being used at the Valencian Regional Ministry of Infrastructure, Territory and Environment.

Keywords: Method Engineering, CAME Environment, Process Support, Model-Driven Engineering

1 Introduction

The definition of a software development method suitable for all situations is now considered unfeasible [9, 12]. For this reason, software organizations need to define (or adapt) their methods in-house in order to meet their specific needs. To help meet this challenge, Method Engineering (ME) research aims to provide solutions [5, 14, 19, 21] to efficiently deal with the definition and adaptation of methods and also with the construction of the supporting software tools.

Similarly to software engineering, which is concerned with all aspects of software and its development, ME is concerned with all aspects of methods and their definition. Thus, most ME approaches define precise engineering solutions that address the definition of the two interrelated aspects that generally comprise methods: product and process [5, 19, 21]. However, while it is commonly

* This work has been developed with the support of MICINN under the project EVERYWARE TIN2010-18011.

agreed that the product aspect of methods represents the artifacts to be produced during the method execution, the process aspect is usually understood in two slightly different ways. Some consider the process aspect as the overall development process of the method, which encompasses all the activity-related issues needed for software development [16]. By contrast, most ME approaches (e.g., [5, 15, 20, 21]) use the term process at a smaller scale, considering a process as the description of how a single method product must be built.

In this work, we consider processes at the greater scale (hence, we denote hereafter the overall process of methods simply as the method *process part*[1]). We argue that supporting the specification and the enactment of the process part of methods brings important benefits. A precise, complete, and well-structured process specification may be useful to facilitate the understanding of how software development is performed within an organization. Furthermore, the enactment of this process specification in a software environment may be useful to guide software engineers throughout the actual development process and also to automate it as far as possible. However, to the best of our knowledge there is no ME approach that supports the specification and the enactment of the process part of methods, and also provides complete software support to these issues.

In order to fill this gap, this paper describes a methodological approach and a supporting software environment that support the specification of the process part of methods and also the construction of the software tools required to enact this process part. The proof of concept is performed in the context of a ME approach and a Computer-Aided Method Engineering (CAME) environment, called MOSKitt4ME[2], that have been presented by the authors in [6, 7]. This ME approach is based on Model-Driven Engineering (MDE) techniques and thereby it proposes defining methods as method models based on the SPEM 2.0 standard [17] (*method design* phase) and semi-automatically building the supporting CASE environments by means of model transformations (*method implementation* phase). This paper focuses on extending both the ME proposal and the MOSKitt4ME tool to enable process specification (during method design) and process enactment (during method implementation). Specifically, the proposal and the CAME environment have been enhanced with an executable process modeling language (BPMN 2.0 [18]) to properly support the specification of the process part of methods. Moreover, the proposal and the CAME environment have also been extended to support the generation of CASE environments that incorporate process enactment through the use of a process engine.

The proposal and the CAME environment presented in this paper are being used in a real case study at the Valencian Regional Ministry of Infrastructure, Territory and Environment, also known as CIT. Specifically, the gvMetrica method has been specified using MOSKitt4ME and a CASE environment has been generated from this specification. Moreover, gvMetrica is being executed in real projects using this CASE environment. The application of the proposal in a real case study allows us to take valuable feedback for improving it.

[1] Likewise, we denote the product aspect of methods as the method *product part*.
[2] MOSKitt4ME is an extension of MOSKitt (http://www.moskitt.org/) for ME.

The paper is structured as follows. First, section 2 presents a brief state-of-the-art review that highlights some of the process support limitations that present current ME approaches. Then, section 3 provides an overview of our proposal. Section 4 describes how the process specification is performed during the method design. Section 5 describes how the process enactment is supported in the method implementation. Finally, section 6 draws some conclusions.

2 State of the Art

The term Method Engineering was first introduced in the mid-eighties by Bergstra *et al.* in [3]. Thereafter, many research efforts have attempted to provide solutions to the challenges that ME entails. Some of the most relevant contributions are those by Brinkkemper [4, 5], Prakash [19, 20], Ralyté [21, 22] and Karlsson [14, 15]. These works are based on a modular view of methods, whereby methods are built by assembling different kinds of methods modules, namely method fragments, method blocks, method chunks and method components respectively. With regard to the process support, these modules focus on the processes necessary to develop specific method products. The method process part is defined when these modules are assembled, generally by means of precedence relationships that establish their execution order. Thereby, the resulting process is quite limited in terms of control flow, since complex behavior (such as the expressed by branching conditions, events, synchronizations, etc.) cannot be defined.

Another important limitation of these approaches refers to process enactment. The proposals by Ralyté [21, 22] and Karlsson [14, 15] do not consider process enactment. Prakash [19, 20] defines an enactment algorithm that is oriented towards the construction of specific method products and, therefore, does not align with process enactment as we intend to support in our ME approach. The support provided for process enactment in Brinkkemper's proposal [4, 5] is more akin to our work. However, since only two types of relationships (i.e., precedence and conditional precedence) can be established between process fragments, the resulting processes, and consequently their enactment, may be too limited for software engineers working on real development projects.

Other important contribution to the ME field is the OPEN Process Framework (OPF) [11]. The OPF provides a repository of method modules (in OPF called method components) that are defined in terms of a meta-model. This meta-model has recently been upgraded to fit the ISO/IEC 24744 standard [13]. While this standard does support the specification of the process part of methods, we still observe some limitations. ISO/IEC 24744 provides limited support for the definition of processes with a complex control flow. In ISO/IEC 24744 process elements are defined as *work units*, which are allocated in *stages*. The control flow of the process is established by the stages, which can only be associated via precedence relationships. Aharoni *et al.* note this problem in [2] and suggest enriching processes by means of an extension of the stage concept that allows work units to be combined within stages in more meaningful ways than simple inclusion (e.g., concurrently or iteratively). Another important limitation refers to the fact that ISO/IEC 24744 does not formalize process execution semantics and therefore process enactment via a process engine is not addressed. What this standard does support is the definition of specific endeavours (enactments) in a project-plan fashion.

Another recent standard initiative is represented by the SPEM 2.0 standard [17]. SPEM 2.0 defines a meta-model for development methods that also presents the problems stated above regarding process (control flow) specification and process enactment. However, SPEM 2.0 presents an important advantage that helps overcome these problems, making it a suitable language to be used in our proposal. Specifically, SPEM 2.0 provides powerful mechanisms for enhancing process definitions via behavioral modeling formalisms such as BPMN 2.0 [18]. This allows method engineers to easily enhance the process definition both in terms of process (control flow) specification and process executability.

To sum up, we consider that the main process support limitations of ME approaches are in terms process control flow specification and enactment. To fill this gap, this paper extends the work presented by the authors in [6, 7]. Our main intent is to meet ME needs regarding support to the process part of methods.

3 Overview of the MOSKitt4ME Approach

Figure 1 shows an overview of our ME approach. The proposal is situated within the context of MDE. Following MDE principles, methods are first defined as models (*method design* phase) and these models are then used by model transformations to generate the supporting CASE environments (*method implementation* phase). During the method design, the method engineer defines both the product and process parts of the method. This is carried out by assembling method fragments that are available in a method base repository [7]. We use the SPEM 2.0 standard for the construction of the method model. Once the method design is finished, a CASE environment that provides support to both the product and process parts of the method is obtained during the method implementation. This is done by means of a Model-To-Text (M2T) transformation.

This paper focuses on the process support provided in both phases of our ME approach. The main goal of the work is to allow method engineers not only to properly specify the process part of methods (during method design) but also to bring this specification to execution (during method implementation). In order to achieve this goal, our proposal has been defined based on a set of needs that must be met. We consider these needs as a first step towards suitable process support in ME and CAME technology.

With respect to method design, we need a language to properly specify the process part of the method. This language must:

– be expressive enough to enable the representation of complete, understandable, unambiguous, and well-structured processes.
– fully formalize process execution semantics so that process enactment support can be provided in the CASE environment supporting the method.

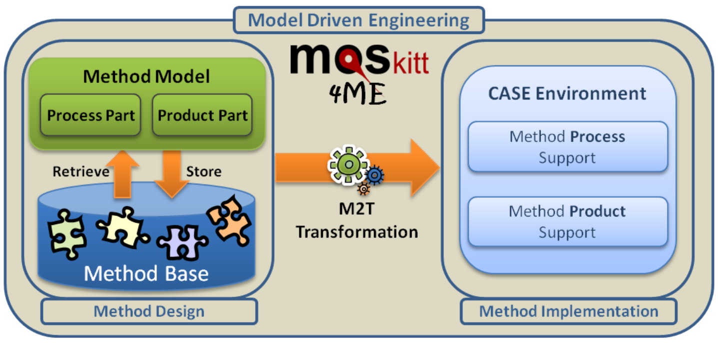


Fig. 1. Overview of MOSKitt4ME

With respect to method implementation, we need to enhance CASE environments with mechanisms that provide support to:

– the execution of the process specification. Process engines provide a set of enactment facilities (such as task orchestration, task automation, constraint enforcement, etc.) that allow CASE environments to provide guidance to software engineers throughout the actual development process and also to partially automate its performance.
– the management of the method products consumed or produced during the process execution. Process engines must be able to invoke the software tools that enable the creation and manipulation of the method products. Therefore, these tools (editors, generators, etc.) must be integrated in the CASE environment supporting the method.

In order to meet the needs regarding method design, our proposal combines the use of SPEM 2.0 with BPMN 2.0 for two main reasons. Firstly, BPMN 2.0 provides more expressiveness than SPEM 2.0 with respect to process specification (e.g., BPMN 2.0 allows defining *events* and *gateways*, which are not supported in SPEM 2.0). Secondly, BPMN 2.0 fully formalizes execution semantics, while SPEM 2.0 is not executable. Thereby, the method is defined in our proposal by means of SPEM 2.0, and the process part is complemented with a BPMN 2.0 model that enhances the process definition in terms of process (control flow) specification and process executability.

In order to meet the needs regarding method implementation, we have developed a software component that is always integrated in the generated CASE environments. This component is built upon a process engine that enables the execution of the process definition by orchestrating method tasks and invoking tools for the development of the method products. In order for the CASE environments to contain the required tools, we make use of reusable assets. These assets are stored in a repository and are associated to method elements during the method design. Then, the M2T transformation takes these assets and integrates them in the generated CASE environment.

4 Process Specification during Method Design

Figure 2 shows our proposal for method design. It is composed of three main steps: *method definition*, *method configuration*, and *executable process definition*. These steps are described below, focusing on how the process specification is performed. Then, subsection 4.1 illustrates these steps with an example.

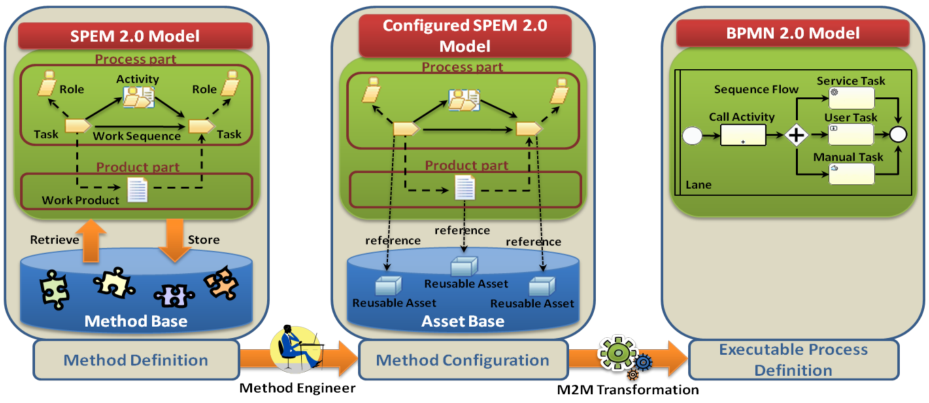


Fig. 2. Method design in MOSKitt4ME

Method Definition. In this step, the method engineer builds the method model by means of SPEM 2.0. As figure 2 illustrates, the main elements that the method engineer must define to build the method process part are *Tasks*, *Activities*, *Work Sequences*, and *Roles*. Tasks represent basic units of work. Activities group tasks and other activities, forming breakdown structures. The root activity of these breakdown structures is named *Delivery Process*. Work sequences represent precedence relationships between tasks and activities. Roles are performers of method tasks. On the other hand, the method product part is composed of *Work Products*, which are inputs and outputs of method tasks.

The construction of this model can be performed by assembling reusable method fragments retrieved from a method base repository. Further details about how these fragments are managed in our ME approach can be found in [6, 7].

Method Configuration. In this step[3], the method engineer defines the tools that will allow software engineers to perform the method tasks during the process enactment, and the guides that will assist them during the tasks performance. To do so, the method engineer links tasks (and work products) with *reusable assets* that are retrieved from an asset base repository. These assets can be *tool*

[3] Note that this phase differs from the ME approach of method configuration [14].

assets, which contain tools (editors, model transformations, etc.) that enable the creation of products, and *guidance assets*, which contain guidelines (textual descriptions, process models, etc.) about the performance of tasks.

The association of reusable assets with tasks and products is performed based on the following observations: for each task, the method engineer must define whether the task is automatic or not. If it is automatic, the task must be associated with a tool asset containing a model transformation. This transformation will be executed when the task is invoked during the process enactment. If no model transformation is associated with the task, then the task is considered as non-automatic. Both automatic and non-automatic tasks can also be associated with a guidance asset. The guidelines contained in these assets will be used as guidance for the user during the process enactment. On the other hand, for each product, the method engineer must define the notation that will be used during the process enactment for the creation of the product. To do so, the method engineer associates the products with tool assets containing either a meta-model or an editor. A meta-model defines the abstract syntax of the notation. An editor defines both the abstract and concrete syntax. In the first case, MOSKitt4ME will provide a default tree-based editor for the creation of the product. In the second case, MOSKitt4ME will provide the full editor contained in the asset.

To summarize, table 1 gathers the associations allowed between method elements and reusable assets. Further details can be found in [6, 7].

Table 1. Associations allowed between method elements and reusable assets

Method Element Type	**Reusable Asset Type**
Task	Guidance Asset
	Tool Asset (Model Transformation)
Work Product	Tool Asset (Meta-model)
	Tool Asset (Editor)

Executable Process Definition. In this step, the method engineer defines an executable representation of the method process part. We have automated this step by means of a Model-to-Model (M2M) transformation that takes the configured SPEM 2.0 model as input and automatically generates a set of BPMN 2.0 processes. Then, these processes can be manually modified to complete the process specification. This is often needed because BPMN 2.0 provides more expresiveness than SPEM 2.0 with respect to process elicitation.

In order to provide more insights on how the transformation obtains the BPMN 2.0 processes, we summarize in table 2 the mappings between the SPEM 2.0 and BPMN 2.0 concepts. The rationale of the mappings is provided below.

1. A SPEM 2.0 *DeliveryProcess* is mapped into a BPMN 2.0 *Process*.
2. A SPEM 2.0 *Activity* is mapped into a BPMN 2.0 *CallActivity* and a BPMN 2.0 *Process*. The *CallActivity* invokes the *Process* when it is executed.

Table 2. Mappings between SPEM 2.0 and BPMN 2.0

	SPEM 2.0	**BPMN 2.0**
1	DeliveryProcess (root Activity)	Process
2	Activity (nested)	Process and CallActivity
3	Task (with ReusableAsset)	ServiceTask
4	Task (with ReusableAsset associated to output WorkProduct)	UserTask
5	Task (without ReusableAsset)	ManualTask
6	WorkSequence	SequenceFlow
7	Role	Lane

3. A SPEM 2.0 *Task* is mapped into a BPMN 2.0 *ServiceTask* if and only if a reusable asset containing a model transformation is associated to the *Task*.
4. A SPEM 2.0 *Task* is mapped into a BPMN 2.0 *UserTask* if and only if the *Task* is not automatic (no reusable asset containing a model transformation is associated to it) and a reusable asset is associated to an output *WorkProduct* of the *Task*.
5. A SPEM 2.0 *Task* is mapped into a BPMN 2.0 *ManualTask* if and only if no reusable asset is associated to the *Task* and its output *WorkProducts*.
6. A SPEM 2.0 *WorkSequence* is mapped into a BPMN 2.0 *SequenceFlow*. The source of the *SequenceFlow* is set to the BPMN 2.0 element generated from the *predecessor* of the *WorkSequence*. The target is set to the BPMN 2.0 element generated from the *successor* of the *WorkSequence*.
7. A SPEM 2.0 *Role* is mapped into a BPMN 2.0 *Lane* if and only if the *Lane* has not been previously generated.

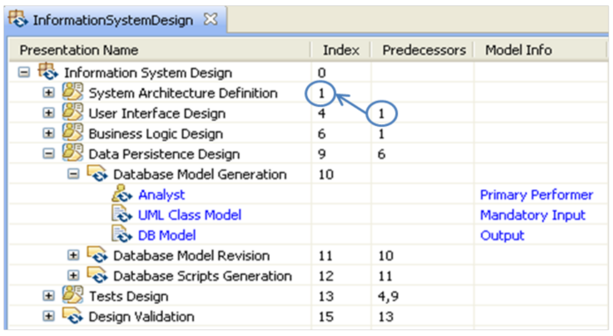
4.1 An Example of Process Specification in MOSKitt4ME

In order to illustrate our proposal for process specification, we present a practical example carried out in MOSKitt4ME. The example process has been modeled in real settings at the CIT. Specifically, it corresponds to an excerpt of gvMetrica that deals with the design of information systems.

Method Definition. This step is performed via the EPF Composer [10], an Eclipse-based editor that has been integrated in MOSKitt4ME to enable the creation of SPEM 2.0 models. Figure 3 shows the example process after being defined by means of this editor. As the figure shows, the first steps of the process are to define the system architecture, the user interface, and the business logic. Then, a model defining the system database schema is obtained from the models specifying the business logic. The database model is then used to automatically obtain the database code. Once all the system artifacts have been created, the test cases can be defined. Finally, the design validation is carried out to validate all the work performed during the process.

As depicted in figure 3, the process is represented in SPEM 2.0 as a breakdown structure that is mainly composed of *Activities* (e.g., *Data Persistence*

Design) and *Tasks* (e.g., *Database Model Generation*), which reference performing *Roles* (e.g., *Analyst*) as well as input and output *Work Products* (e.g., *UML Class Model*). Moreover, there are *Work Sequences* that are established between method elements (e.g., *System Architecture Definition* is a predecessor of *User Interface Design*).


InformationSystemDesign

Presentation Name	Index	Predecessors	Model Info
Information System Design	0		
System Architecture Definition	1		
User Interface Design	4	1	
Business Logic Design	6	1	
Data Persistence Design	9	6	
Database Model Generation	10		
Analyst			Primary Performer
UML Class Model			Mandatory Input
DB Model			Output
Database Model Revision	11	10	
Database Scripts Generation	12	11	
Tests Design	13	4,9	
Design Validation	15	13	

Fig. 3. Example process in SPEM 2.0

Method Configuration. This step is performed via a repository client that allows method engineers to retrieve reusable assets from the asset base [7]. As an example, let us consider the task *Database Model Generation*. The execution of this task obtains a database model from a UML class model. To specify this behavior, the method engineer can associate the task with an asset containing the Eclipse plug-ins that implement the UML2DB transformation provided by the MOSKitt tool. On the other hand, the work products of the task can be associated with assets containing the UML meta-model and the MOSKitt SQLSchema meta-model respectively.

Executable Process Definition. This step is performed via the M2M transformation described above, which has been implemented in MOSKitt4ME as an extension of the EPF Composer. Figure 4 shows the BPMN 2.0 processes resulting from applying this transformation to the example process. These processes are represented in terms of the Activiti Designer [1], an Eclipse-based graphical editor that has been integrated in MOSKitt4ME to support BPMN 2.0.

To illustrate how the processes shown in figure 4 have been generated, we present below some examples of the application of the mappings of table 2.

1. The SPEM 2.0 *Delivery Process* "*Information System Design*" is mapped into a BPMN 2.0 *Process* (diagram "*A*" in figure 4).
2. The SPEM 2.0 *Activity* "*Data Persistence Design*" is mapped into a *Call Activity* and a BPMN 2.0 *Process* (diagram "*B*" in figure 4).

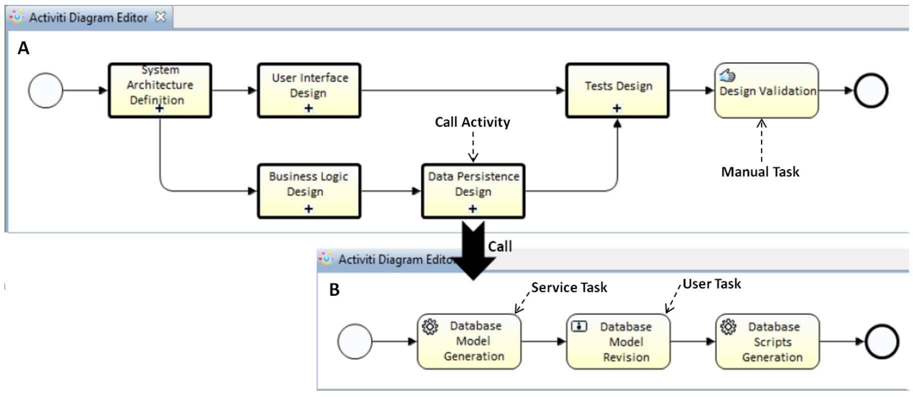


Fig. 4. Generated BPMN 2.0 processes

3. The SPEM 2.0 *Task* "*Database Model Generation*" is mapped into a *Service Task* since it has a M2M transformation associated to it as a reusable asset.
4. The SPEM 2.0 *Task* "*Database Model Revision*" is mapped into a *User Task* since it is not automatic but has an output product with a reusable asset associated to it (not shown in the example).
5. The SPEM 2.0 *Task* "*Design Validation*" is mapped into a *Manual Task* since it does not have any reusable asset associated to it.
6. The SPEM 2.0 *Work Sequences* are mapped into the *Sequence Flows* that connect the BPMN 2.0 elements in both diagrams.

After the generation of the BPMN 2.0 processes, the method engineer can manually modify them to specify more complex control flows. For instance, the method engineer can add a *Gateway* to specify that the *Call Activity* "*Tests Design*" must not be executable until all its predecessors are finished.

5 Process Enactment during Method Implementation

The method implementation phase is in charge of the construction of the CASE tool support for the method specified during the method design. In MOSKitt4ME, CASE environments are semi-automatically built by means of a M2T transformation. We provide details about this transformation in [8].

In this section we focus on how CASE environments are structured to support the process enactment. This structure is depicted in figure 5. As this figure shows, CASE environments are divided into three main parts: the components that provide *method product support*, the components that provide *method process support*, and the *Project Manager Component* (PMC). These parts are introduced below. Subsection 5.1 illustrates them with an example of process enactment.

Method Product Support. The software components that provide product support must enable the creation and manipulation of the method products. In

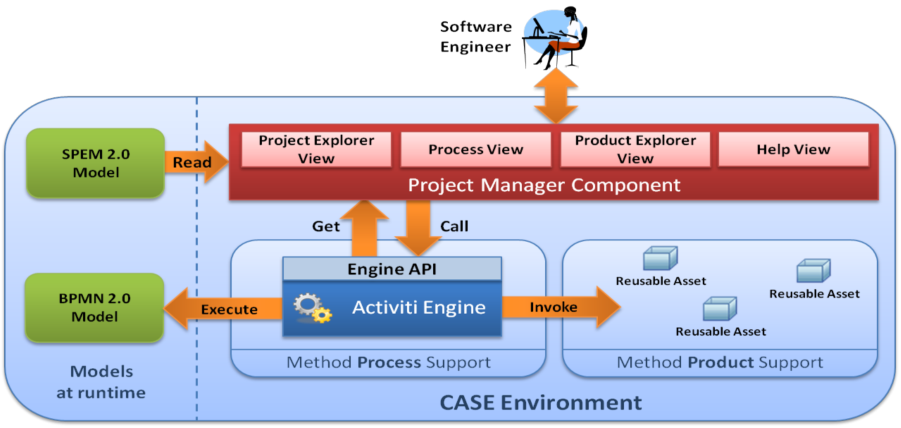


Fig. 5. Method implementation in MOSKitt4ME

MOSKitt4ME, these components correspond to the reusable assets associated to the method elements during the method configuration. Specifically, *Tool assets* allow software engineers to create the method products by means of model transformations or software applications such as graphical or textual editors. *Guidance assets* do not allow software engineers to directly create the method products, but they do provide guidance on how the method tasks must be performed to properly develop these products.

Method Process Support. The software components that provide process support must enable the execution of the BPMN 2.0 model. In MOSKitt4ME, this functionality is provided by the Activiti Engine [1]. A significant feature of this engine is that it supports a lot of different task types. Specifically, the behavior of the engine is the following:

– *Service tasks*: when a service task becomes active, it is automatically executed. The execution of a service task invokes the model transformation that is associated to the task as a reusable asset.
– *User tasks*: when a user task becomes active, the engine invokes the software tools that enable the creation of the output products of the task. These tools are associated to the products as reusable assets.
– *Manual tasks*: when a manual task becomes active, the engine does not perform any action.
– *Call activities*: when a call activity becomes active, the engine automatically starts a new instance of the BPMN 2.0 process referenced by the call activity.

Project Manager Component. The PMC provides a graphical user interface for the CASE environment and assists software engineers during the process enactment (i.e., during the course of the development projects). To achieve this goal, it makes use of the Activiti engine to execute BPMN 2.0 process instances and also makes use of the SPEM 2.0 model to extract information about the method that is not represented in the BPMN 2.0 model (since this model only contains the process part). The PMC is divided into the following Eclipse views:

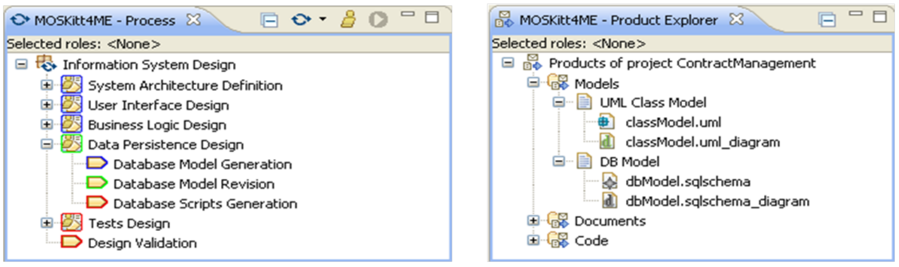
– *Project Explorer*: This view is provided as part of the Eclipse platform and has been integrated in the PMC to show the projects that have been created and the resources they contain (files, folders, etc.). From this view, the software engineer can create new projects, delete existing projects, etc.
– *Process*: This view shows the current state of the process instance associated to the project that is selected in the Project Explorer view. From this view, the software engineer can invoke the execution of the tasks that are executable. Once a task is finished, the PMC invokes the engine API to set the task as executed and proceed to the next state of the process. This view also offers the possibility to filter the tasks based on the role of the users so that they only see the tasks that they are assigned to.
– *Product Explorer*: This view shows a hierarchical picture of the artifacts that have been produced during the course of the project that is selected in the Project Explorer view. This hierarchy is based on domains, subdomains, and work product elements, which are read from the SPEM 2.0 model. This view also offers the possibility to filter the information based on the role of the users so that they only see the artifacts that they are responsible for.
– *Help*: This view is provided as part of Eclipse and has been integrated in the PMC to provide guidance to software engineers during the performance of the method tasks. This view is dynamically updated based on the task that is selected in the Process view. The guides to show are known by the PMC because they were associated to the selected task as a reusable asset.

5.1 An Example of Process Enactment in MOSKitt4ME

In order to illustrate how processes are enacted in MOSKitt4ME, we continue with the example introduced in subsection 4.1. Let us consider that a new project has been created by means of the Project Explorer view. When this project is selected, the Process and Product Explorer views are updated accordingly. At this point, the Process view shows the initial state of the process and the Product Explorer view is empty. Now, let us consider the *System Architecture Definition*, *User Interface Design*, and *Business Logic Design* activities have been executed, and the next executable task is *Database Model Generation.* Since this task is automatic, the PMC automatically invokes the associated transformation. Once the transformation is finished, the task is set as executed and the process proceeds to the next state. This state is illustrated in figure 6.

The Process view (left)[4] depicts the current state of the process instance. Specifically, this view shows the tasks and activities in different colors depending

[4] Additional screenshots of the tool can be found in [6–8] and also in https://users.dsic.upv.es/∼vtorres/moskitt4me/


MOSKitt4ME - Process
Selected roles: <None>
- Information System Design
  - System Architecture Definition
  - User Interface Design
  - Business Logic Design
  - Data Persistence Design
    - Database Model Generation
    - Database Model Revision
    - Database Scripts Generation
  - Tests Design
  - Design Validation

MOSKitt4ME - Product Explorer
Selected roles: <None>
- Products of project ContractManagement
  - Models
    - UML Class Model
      - classModel.uml
      - classModel.uml_diagram
    - DB Model
      - dbModel.sqlschema
      - dbModel.sqlschema_diagram
  - Documents
  - Code

Fig. 6. Process and Product Explorer views

on whether they have already been executed (blue), are executable (green), or are not executable (red) in the current state of the process. Note that, even though the Process view shows the process in terms of SPEM 2.0, the process instance corresponds to an instance of a BPMN 2.0 process. This is possible because there is a one-to-one correspondence between SPEM 2.0 tasks and BPMN 2.0 tasks.

The Product Explorer view (right) depicts some artifacts that have been produced during the execution of the process. In this case, it is showing the input and output work product elements of the last executed task (i.e., *Database Model Generation*). As the figure shows, work product elements are categorized by domain and contain the actual products, that is, the files.

Now, let us consider that the user wants to proceed with the process execution. To do this, the user selects the task *Database Model Revision* in the Process view. This action has a twofold effect. The Help view is updated to show textual guidance about the task and, since the task is a user task, the PMC opens the software tool that allows the software engineer to carry out the task.

Once all the tasks have been executed, the process engine deletes the process instance and, therefore, the project can be considered as concluded.

6 Conclusions

In this paper we present an extension of our ME approach and CAME environment (MOSKitt4ME) [6, 7] so as to provide adequate support to process specification during method design and process enactment during method implementation. This extension builds on the idea of combining the use of SPEM 2.0 and BPMN 2.0. This is based on the fact that BPMN 2.0 can resolve SPEM 2.0 limitations with respect to process elicitation and process executability.

To validate of the proposal, MOSKitt4ME is currently being used in real settings at the CIT. This is providing us initial feedback that is allowing us to identify limitations of our work. For instance, MOSKitt4ME does not yet properly deal with the dynamic nature of projects. Therefore, we are concerned with supporting variability and evolution of methods at runtime. So, we will be soon working on providing mechanisms to meet these challenges. Specifically, we propose the introduction in MOSKitt4ME of a reconfiguration engine that enables the CASE tool reconfiguration at runtime based on context changes.

References

1. Activiti: http://www.activiti.org/
2. Aharoni, A., Reinhartz-Berger, I.: A domain engineering approach for situational method engineering. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008, LNCS, vol. 5231, pp. 455–468. Springer Berlin / Heidelberg (2008)
3. Bergstra, J., Jonkers, H., Obbink, J.: A software development model for method engineering. In: Esprit 1984: Status Report of Ongoing Work (1985)
4. Brinkkemper, S.: Method engineering: Engineering of information systems development methods and tools. Information and Software Technology 38, 275–280 (1996)
5. Brinkkemper, S., Saeki, M., Harmsen, F.: Meta-modelling based assembly techniques for situational method engineering. Inf. Syst. 24, 209–228 (1999)
6. Cervera, M., Albert, M., Torres, V., Pelechano, V.: A methodological framework and software infrastructure for the construction of software production methods. In: Yang, Y., Münch, J., Schäfer, W. (eds.) ICSP 2010. LNCS, vol. 6195, pp. 112–125. Springer-Verlag, Berlin, Heidelberg (2010)
7. Cervera, M., Albert, M., Torres, V., Pelechano, V.: Turning method engineering support into reality. In: Ralyté, J., Mirbel, I., Deneckère, R. (eds.) ME 2011, IFIP AICT, vol. 351, pp. 138–152. Springer Boston (2011)
8. Cervera, M., Albert, M., Torres, V., Pelechano, V., Bonet, B., Cano, J.: A technological framework to support model driven method engineering. In: Actas de los Talleres de las JISBD. pp. 47–56 (2010)
9. Cockburn, A.: Selecting a project's methodology. IEEE Software 17, 64–71 (2000)
10. Eclipse Process Framework: http://www.eclipse.org/epf/
11. Firesmith, D., Henderson-Sellers, B.: The OPEN Process Framework: An Introduction. Addison-Wesley (2002)
12. Henderson-Sellers, B., Ralyté, J.: Situational method engineering: State-of-the-art review. J. UCS. 16, 424–478 (2010)
13. ISO/IEC: Software Engineering: Metamodel for Development Methodologies. ISO/IEC 24744 (2007)
14. Karlsson, F., Ågerfalk, P.J.: Method configuration: adapting to situational characteristics while creating reusable assets. Information and Software Technology 46, 619–633 (2004)
15. Karlsson, F., Ågerfalk, P.J.: Towards structured flexibility in information systems development: Devising a method for method configuration. J. Database Manag. 20, 51–75 (2009)
16. Niknafs, A., Asadi, M.: Towards a process modeling language for method engineering support. In: 2009 WRI World Congress on Computer Science and Information Engineering. vol. 07, pp. 674–681. IEEE Computer Society (2009)
17. OMG: Software & Systems Process Engineering Metamodel (v2.0) (2007)
18. OMG: Business Process Model and Notation (v2.0) (2011)
19. Prakash, N.: Towards a formal definition of methods. Requir. Eng. 2, 23–50 (1997)
20. Prakash, N.: On method statics and dynamics. Inf. Syst. 24, 613–637 (1999)
21. Ralyté, J., Rolland, C.: An approach for method reengineering. In: S.Kunii, H., Jajodia, S., Sølvberg, A. (eds.) ER 2001, LNCS, vol. 2224, pp. 471–484. Springer Berlin / Heidelberg (2001)
22. Ralyté, J., Rolland, C.: An assembly process model for method engineering. In: Dittrich, K., Geppert, A., Norrie, M. (eds.) Advanced Information Systems Engineering, LNCS, vol. 2068, pp. 267–283. Springer Berlin / Heidelberg (2001)