Model Driven Method Engineering: a Case Study

Mario Cervera, Manoli Albert, Victoria Torres, Vicente Pelechano
Chapter 5
A Case Study

This chapter presents a case study that has been developed to validate the methodological framework and the software architecture proposed in this thesis. In this case study MOSKitt4ME has been used for specifying a software production method and generating its supporting CASE environment. In particular, the software production method comes from [Torres08]. This method defines an MDD approach for the generation of web applications supporting business process specifications.

This chapter is structured as follows: first, section 5.1 provides an overview of the case study. Then, section 5.2 describes in detail how it has been developed in MOSKitt4ME. Finally, section 5.3 outlines some conclusions.

5.1. The OOWS-BP method

OOWS-BP [Torres08] is a software production method that results from extending the OOWS web engineering approach [Fons08]. This extension introduces and modifies some of the existing steps in order to deal with the execution of business processes. Thus, OOWS-BP embodies an MDD approach for the generation of business process-driven web applications from conceptual models. Briefly presented, the OOWS-BP method (see Fig. 5.1) involves the participation of three different roles: the analyst and the developer (related to human beings), and the buzzy tool (which represents the software system). The process is started by the analyst who specifies, by means of the BPMN notation, the business processes that have to be supported in the web application. This specification constitutes a non-executable version of the process models, which require more details to be deployed and run in a process engine. Then, in the next step, the developer performs the system specification, i.e. the business process is defined in terms
of the OO-Method models [Pastor01] and the OOWS services model. Once
the system specification is finished, these models are used by model-to-model
(M2M) transformations that generate the OOWS navigational and
presentation models, and the business process in WS-BPEL (an executable
representation of the business process). Finally, the Tapestry\(^1\) files that
implement the web application are obtained from the OOWS models (which
can be manually modified by the developer).

Fig 5.1. The OOWS-BP method

\(^1\) Tapestry, http://tapestry.apache.org/
5.2. Development of the case study

This section details how the case study has been developed in MOSKitt4ME following the methodological framework presented in chapter 3. Specifically, the section is divided into three subsections (based on the framework phases) in order to describe how the model of the method is built and how the supporting CASE tool is obtained from this model.

5.2.1. Method design

In this phase, the EPF Composer editor is used for the creation of the method model. Following the process defined in chapter 3 (paradigm-based approach), the method model is created in two steps, (1) the definition of the product model and (2) the definition of the process model.

In this proposal, the product model and the method content part of a SPEM method are considered analogous, therefore, the first step has been to create by means of the EPF Composer the method content of the OOWS-BP method. Since at this stage the method model is specified without detailing the techniques, languages and notations that will be used during the method enactment, this part of the model is composed of generic products (e.g. business process model, services model, etc.), tasks (e.g. business process analysis, system specification, etc.) and roles (e.g. analyst, developer, etc.).

Once the method content is defined, the process model is built. The process model corresponds to the method process part of a SPEM method. Therefore, the second step has been to create by means of the EPF Composer the Work Breakdown Structure that establishes the tasks execution order.

Furthermore, as described in chapter 3, during the construction of the method model it is possible to reuse conceptual fragments stored in the Method Base repository (assembly-based approach). Specifically, during the definition of the OOWS-BP method model a product fragment containing the task system specification has been used. In order to do so, first it has been extracted from the repository by means of the repository client, and then, its content (i.e. the method task) has been automatically integrated into the method content part of the model.
Fig. 5.2 shows a snapshot of the EPF Composer containing the OOWS-BP method model resulting from the method design phase. On the left part of the figure, the Library view shows some method content elements (i.e. tasks, roles, etc.) in a tree viewer. On the right part, details of the process are depicted as a Work Breakdown Structure.

Moreover, table 5.1 provides further details about all these tasks. Specifically, this table contains the tasks predecessors, the performing roles and the input/output products. In addition, all the tasks are briefly described below.
Table 5.1. OOWS-BP tasks

**Business Process Analysis**

The analyst specifies as a non-executable process model the business process that will be supported by the generated web application.

**System Specification**

The developer defines the business process in terms of the OO-Method models and the services model.

**Business Process Design (subprocess)**

The developer completes the business process model with additional information.

**Business Process Model Preprocess**

The developer builds an extension of the business process model in order to specify additional information that is not supported by the notation used to create the business process model.

**Business Process Design**
The developer completes the business process model with information that was not specified by the analyst.

**Web Specification**

This task automatically generates from the previously built models a navigational model and a presentation model, that is, the specification of the web application as defined by the OOWS approach [Fons08].

**Web Application Generation**

This task automatically generates the web application from its specification. This application is implemented by means of the framework Tapestry.

**Executable Business Process (subprocess)**

This task embodies a transformation chain that obtains the executable WS-BPEL specification from the business process model.

*BPMN to Babel*

This automatic task executes a M2M transformation that obtains an intermediate representation of the business process model (babel notation).

*Babel to BPEL*

This automatic task executes a M2M transformation that transforms the business process model (in babel notation) into an executable WS_BPEL model.

*WS-BPEL Completion*

This task executes a M2M transformation that completes the WS-BPEL model so that it can be imported by the process engine.

*WSDL and XSD Generation*

This task generates the WSDL and XSD files that complete the WS-BPEL model in order to make it deployable. Specifically, the WSDL files define the interface associated to the new service defined by the WS-BPEL and the XSD files define the data types used by it.
5.2.2. Method configuration

Once the product and process parts of the method model have been specified, the method configuration phase can start. Following the process defined in chapter 3 for method configuration, in this phase the method engineer must make use of the repository client in order to (1) select technical fragments and (2) associate them with tasks and products of the method model. This association represents that the technical fragments must be included in the generated CASE tool in order to provide support to the tasks and products they are associated to.

Fig. 5.3 shows the Eclipse view that implements the repository client in MOSKitt4ME. This view is showing the fragments that support the OOWS-BP tasks and products. Moreover, the right part of Fig. 5.3 shows an example of association of a technical fragment (BPMN editor) with a method product (business process model).

In order to give more information about all these fragments, table 5.2 shows for each of them the supported method elements and the Eclipse plugins it contains. In addition, a brief description of the fragments is given.
below. Finally, the overall associations between the OOWS-BP elements and the technical fragments are summarized in tables 5.3 and 5.4.

<table>
<thead>
<tr>
<th>Technical fragment</th>
<th>Supported element</th>
<th>Plugin names</th>
</tr>
</thead>
<tbody>
<tr>
<td>OOWS metamodel</td>
<td>OOWS model¹, OOWS model²</td>
<td>oowsModel, oowsModel.edit, oowsModel.editor, oowsModel.feature</td>
</tr>
<tr>
<td>BPMNX metamodel</td>
<td>Business process model extension</td>
<td>bpmnxModel, bpmnxModel.edit, bpmnxModel.editor, bpmnxModel.feature</td>
</tr>
<tr>
<td>BPMN2OOWS transformation</td>
<td>Web Specification</td>
<td>bpmn2oows, bpmn2oows.feature</td>
</tr>
<tr>
<td>OOWS2WebApplication transformation</td>
<td>Web Application Generation</td>
<td>oows2webApplication, oows2webApplication.feature</td>
</tr>
<tr>
<td>Babel metamodel</td>
<td>Babel model</td>
<td>babelModel, babelModel.edit, babelModel.editor, babelModel.feature</td>
</tr>
<tr>
<td>Bpmn2babel transformation</td>
<td>BPMN to Babel</td>
<td>bpmn2babel, bpmn2babel.feature</td>
</tr>
<tr>
<td>Babel2bpe1 transformation</td>
<td>Babel to BPEL</td>
<td>babel2bpe1, babel2bpe1.feature</td>
</tr>
<tr>
<td>BPEL refinement transformation</td>
<td>WS-BPEL Completion</td>
<td>bpe1Refinement, bpe1Refinement.feature</td>
</tr>
<tr>
<td>OOWS2WSdlAndxsd transformation</td>
<td>WSDL and XSD generation</td>
<td>oows2wsdlAndxsd, oows2wsdlAndxsd.feature</td>
</tr>
</tbody>
</table>

¹ OO-method models and services model
² Modified
³ Updated with navigational and presentation models

Table 5.2. OOWS-BP technical fragments

**BPMN editor (STP)**

This technical fragment contains the Eclipse plugins that implement the BPMN graphical editor developed as part of the SOA Tools Platform Project (STP) [STP].

**OOWS metamodel**

This fragment contains the plugins that implement the OOWS metamodel.
BPMNX metamodel

This fragment contains the plugins that implement the extension of the BPMN metamodel.

BPMN2OOWS transformation

This technical fragment encapsulates the M2M transformation implemented in ATL [ATL] that obtains the OOWS navigational and presentation models.

OOWS2WebApplication transformation

This fragment encapsulates the M2T transformation implemented in MOFScript [MOFScript] that obtains the final web application from the conceptual models (OOWS, BPMN, etc.).

Babel metamodel

This fragment contains the plugins that implement the Babel metamodel. This metamodel enables the creation of BPMN models that can be transformed in WS-BPEL models by the transformation bpmn2bpel.

Bpmn2babel transformation

This fragment contains the plugins that implement the ATL M2M transformation that automatically obtains a babel model from a business process model specified in BPMN.

Babel2bpel transformation

This fragment contains the M2M transformation implemented in ATL that obtains the WS-BPEL model from the Babel model.

BPEL refinement transformation

This fragment encapsulates the plugins that implement the ATL M2M transformation that completes the WS-BPEL model so that it can be imported in the process engine.

OOWS2WSDLandXSD transformation
This fragment contains the M2T transformation implemented in MOFScript that obtains the WSDL and XSD files associated to the WS-BPEL process.

**Summary**

In order to provide an overview of the products and task of the method and which technical fragments provide support to them, table 5.3 shows the associations between products and technical fragments and table 5.4 the associations between tasks and technical fragments.

<table>
<thead>
<tr>
<th>Method Product</th>
<th>Technical Fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business process model</td>
<td>BPMN editor (STP)</td>
</tr>
<tr>
<td>OOWS model(^1)</td>
<td>OOWS metamodel</td>
</tr>
<tr>
<td>Business process model extension</td>
<td>BPMNX metamodel</td>
</tr>
<tr>
<td>Business process model(^2)</td>
<td>BPMN editor (STP)</td>
</tr>
<tr>
<td>OOWS model(^3)</td>
<td>OOWS metamodel</td>
</tr>
<tr>
<td>Tapestry files</td>
<td>No fragment has been specified</td>
</tr>
<tr>
<td>Babel model</td>
<td>Babel metamodel</td>
</tr>
<tr>
<td>WS-BPEL model</td>
<td>No fragment has been specified</td>
</tr>
<tr>
<td>WS-BPEL model(^4)</td>
<td>No fragment has been specified</td>
</tr>
<tr>
<td>WSDL and XSD</td>
<td>No fragment has been specified</td>
</tr>
</tbody>
</table>

\(^1\) OO-method models and services model
\(^2\) Modified
\(^3\) Updated with navigational and presentation models
\(^4\) Completed

**Table 5.3. Relationship between method products and technical fragments**

<table>
<thead>
<tr>
<th>Method Task</th>
<th>Technical Fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Process Analysis</td>
<td>Guide (optional)</td>
</tr>
<tr>
<td>System Specification</td>
<td>Guide (optional)</td>
</tr>
<tr>
<td>Business Process Model Preprocess</td>
<td>Guide (optional)</td>
</tr>
<tr>
<td>Business Process Design</td>
<td>Guide (optional)</td>
</tr>
<tr>
<td>Web Specification</td>
<td>BPMN2OOWS transformation</td>
</tr>
<tr>
<td>Web Application Generation</td>
<td>OOWS2WebApplication transformation</td>
</tr>
<tr>
<td>BPMN to Babel</td>
<td>Bpmn2babel transformation</td>
</tr>
<tr>
<td>Babel to BPEL</td>
<td>Babel2bpel transformation</td>
</tr>
<tr>
<td>WS-BPEL Completion</td>
<td>BPEL refinement transformation</td>
</tr>
<tr>
<td>WSDL and XSD Generation</td>
<td>OOWS2WSDLandXSD transformation</td>
</tr>
</tbody>
</table>

**Table 5.4. Relationship between method tasks and technical fragments**
5.2.3. Method implementation

In this phase, the method engineer invokes the model transformation that obtains from the configured method model the CASE tool that supports the method. As described in chapter 4, this transformation has been implemented in MOSKitt4ME as a M2T transformation that obtains from the method model a product configuration file through which the final tool is obtained. This tool is a MOSKitt reconfiguration that only contains the required plugins to support the method (i.e. the plugins contained in the technical fragments, the process engine and the Eclipse views that compose the GUI). Specifically, the M2T transformation is invoked by means of the MOSKitt transformation manager, which is shown in Fig. 5.4. Through this Eclipse view, all the transformations registered in MOSKitt can be launched.

![MOSKitt Transforms](image)

**Fig. 5.4.** MOSKitt transformation manager

When selecting the *SPEM2MOSKittConf* transformation, a wizard is opened (see Fig. 5.5). Specifically, in this wizard the input and output parameters of the transformation can be specified. The input parameter corresponds to the SPEM model resulting from the method configuration phase. The output parameter corresponds to the product configuration file through which the MOSKitt reconfiguration supporting the method will be obtained.
Once the product configuration file is generated, the export wizard is automatically launched. This wizard is shown in Fig. 5.6. Specifically, it allows the method engineer to generate the final CASE tool from the product configuration file. For this purpose, at least the following information must be specified:

- **Configuration**: The product configuration file. It is automatically set when the wizard is opened.
- **Root directory**: Name of the folder hosting the generated tool. By default this folder is named `eclipse`.
- **Destination**: Path of the file system where the folder `Root directory` will be placed. If selected the option “Archive file” a package (zip file) of the tool is obtained.

![Fig. 5.5. MOSKitt transformation wizard](image)
In order to illustrate the contents of the product configuration file, Fig 5.7 shows the list of features that establish the plugins that must be included in the final CASE tool. Specifically, the features that correspond to the plugins contained in the technical fragments have been emphasized. The remaining features correspond to software dependencies and to the Project Manager Component.
Once the export process is finished, a MOSKitt reconfiguration supporting the method is obtained. As already shown in chapter 4, these generated tools include (aside from the Eclipse plugins that support the method) a software component called the *Project Manager Component*. This component provides a series of Eclipse views that assist the software engineer during the method enactment. As an example, Fig. 5.8 and Fig. 5.9 show snapshots of the *Product Explorer* and *Process* views respectively. Specifically, in the example, the state of the running method instance is on the *system specification* task. Therefore, the *Product Explorer* view only shows the product *business process model*, and the *Process* view only shows as executable the task *system specification*. 
5.3. Conclusions

The methodological framework and software architecture proposed in this thesis have proven successful in supporting the design and implementation of the OOWS-BP method [Torres08]. This chapter illustrates how the three phases that compose the framework have been followed to build the method specification and to obtain a CASE tool that supports the method.
It is worth noting that the CASE tool that has been obtained provides rich support to the method, since it integrates in a seamless way all the tools that are required to support its execution and also provides guidance to the software engineer in the performance of the method tasks. For a more in-depth view on how the case study has been developed, snapshots and screencasts are available at http://users.dsic.upv.es/~vtorres/moskitt4me/.
References


