Collaborative Modeling of Web Applications for Various Stakeholders

Martin Vasko, Ernst Oberortner, and Schahram Dustdar
{m.vasko|e.oberortner|dustdar@infosys.tuwien.ac.at}

Technical University of Vienna
Distributed Systems Group
Argentinierstraße 8/184-1
A-1040 Vienna
Austria

Abstract The development of Web applications involves many stakeholders with different expertise. In many cases, the development of Web applications provides only technical models and their implementations, making it hard for non-technical stakeholders to get involved into the development process. As a result, a permanent collaboration and communication between all participating technical and non-technical stakeholders is needed during the whole development process. This paper proposes a model-driven approach of Domain-specific Modeling Languages (DSML) to separate the technical details from the non-technical ones. Non-technical stakeholders, also called domain experts, can assist the technical experts to map not well-known domain problems to the appropriate technological models. This leads to an intense collaboration between the different stakeholders and lowers the possibility of misunderstandings. These concepts improve the collaboration and, as a consequence, lowers the possibility of misunderstandings. Also, the model-driven approach leads to a better maintainability and reusability of the resulting Web application.

1 Introduction

A major requirement of developing Web applications is to involve all different types of stakeholders – technical and non-technical – within the development process [1,2]. Because of the complexity of the development process and the Web applications itself, the simplest Web application development teams require time consuming interactions for negotiating the required functionalities of the Web application. Different background and expertise of the involved stakeholders complicates the communication and the collaboration within the design and development process of Web applications.

To reduce the development time, to enhance the maintenance, and to involve all stakeholders, the Model-driven Software Development (MDSD) paradigm can be used [3]. A possible and convenient way for modeling and developing software, especially Web applications, is to use model-driven Domain-specific Modeling

Languages (DSML) [4]. DSMLs are small languages that are tailored to be particularly expressive in a certain problem domain. A DSML describes the domain knowledge via a graphical or textual syntax which is tied to domain-specific modeling elements. Nowadays, DSMLs are often developed by following the MDSD paradigm to describe the graphical or textual syntax through a precisely specified language model. Using MDSD-based DSMLs for modeling Web applications enables technical and non-technical experts to work at higher levels of abstraction to be independent of the underlying technologies [3].

In this work, we introduce an approach to improve the collaborative development of Web applications. Each stakeholder can work with a DSML which is tailored for the background knowledge and the responsibility. Non-technical experts, from now on called domain experts, can work in a higher level of abstraction without using technical aspects, such as Application Programming Interface (API) calls or XML configuration files. Technical experts are able to map not well-known domain problems to an appropriate technological model which provides the concepts of the underlying technology. This abstraction concept significantly improves collaboration between technical and domain experts. Beside the improved collaboration between the involved experts the separation into different models leads to better maintainability and introduces better reuse possibilities for the resulting Web Application.

This work is structured as follows: Section 2 introduces the development process of a Web application to exemplify our approach. The sample Web application outlines the involvement of different stakeholders and introduces the contribution of our approach. Section 3 motivates the consequent usage of MDSD based DSMLs in the domain of Web application development. Section 4 gives a comprehensible overview of our approach. Section 5 applies the approach on the motivating example and outlines the key contributions. A web-based designer is presented to enable a collaborative modeling of the pageflow of Web applications. Section 6 lists some benefits and drawbacks of our approach which were collected during this work. In Section 7 we relate our approach to existing approaches and outline the differences to related projects. Finally, Section 8 summarizes and concludes the paper.

2 Motivating Example

To illustrate the principles of the approach presented in this work, this section introduces a sample scenario. Consider a medium sized enterprise planning to establish a customer management Web application. Assume the following team structure:

- The *manager* initiates the establishment of a software application to enable customer management. We assume that company-wide requirements, such as location-independency or secure access, are also determined by the manager.
- *Project leaders* are responsible for the successful implementation, for the adaptation of the required tasks to the project team members, and for the co-

ordination of the maintenance of the required customer management software application.

– *Project members* perform the daily tasks and request relevant customer information from the software application. Furthermore, project members maintain the customer information by updating customer contact information and tracking customer requests.
– *Technical experts* have detailed knowledge about the existing technology within the enterprise, such as the Web servers, the software frameworks, the database management systems, or the operating systems. Furthermore, technical experts are responsible to deploy, administrate, and maintain the required software applications by the domain experts.

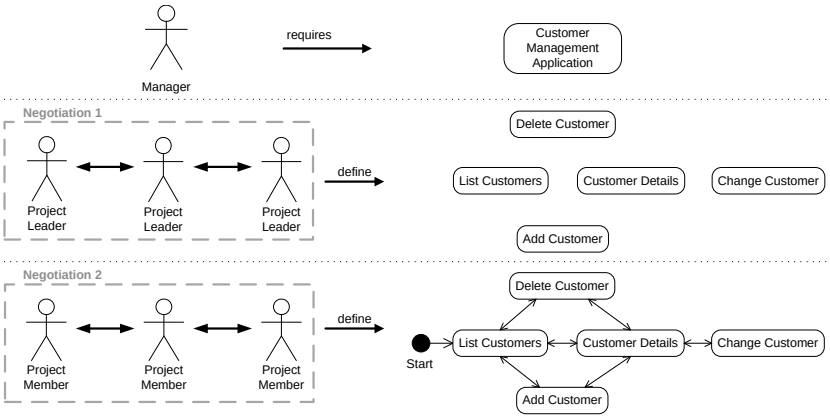

Figure 1. The different stakeholders and their responsibilities

The manager tells the requirements for the software application to the project leader. After negotiating the functionality of the software application, the manager informs the technical expert about the new project. The technical expert suggests a web-based solution to meet the requirements of location-independency and secure access for employees. A Web application simplifies the access to the customer data for the employees and has the lowest installation requirements on the client systems. The low installation requirements of web-based solutions enable a consistent access to remote customer advisers.

The project leader maps the requirements for administrating customers to the concrete functions *Add Customer*, *Change Customer*, and *Delete Customer*. Beside this basic functionality the project leader discusses with leaders of other teams the functionality to access and update all relevant customer data. The project leaders agree upon the functions *List Customer*s to enable access to all customers and *Change Customer* to update customer details. The project leader negotiates these functionalities with the project member to concretize the sequence of functions. Figure 1 refers to this interaction as *Negotiation 1*.

The resulting functionalities are propagated to the project members of each project team. As the members need to interact with the customer management system in their daily work they ascertain the pageflow of the final Web application. This interaction is illustrated in Figure 1 *Negotiation 2*.

The required functionalities of the software application, such as *Add Customer*, *Change Customer*, and *Delete Customer*, do not include technological details at an abstraction level with is familiar to the domain experts. Hence, the requirements of a software application are called the domain concepts. To develop an executable software application which fulfills all needed requirements, the domain concepts have to be mapped to the technical concepts of the existing enterprise's technologies.

Mostly, domain experts, such as managers, project leaders, or project members, do not have any knowledge about the existing technologies. Hence, domain experts have to interact and collaborate with technical experts permanently. Figure 2 depicts the interaction structure of the motivating example. Managers interact with project leaders and project leaders interact with their project members.

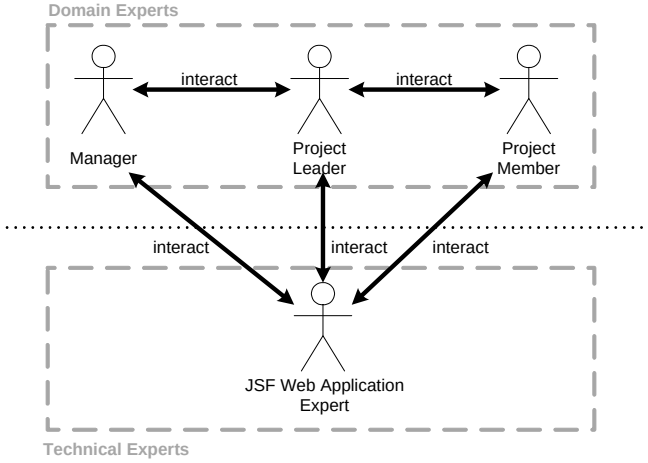

Figure 2. Interaction between the different Stakeholders

The presented procedure of defining the functionalities of the required customer management Web application is fragmented into different stakeholders. Every group of participants enriches the existing model by specific information and expertise. The group of project leaders negotiates the functionalities and the group of project members ascertains the pageflow of the Web application. Technical experts enrich the requirements with technical requirements for the resulting executable Web application.

Due to the diverse backgrounds and knowledge of the different stakeholders, it makes sense to present to each group of stakeholders only the familiar concepts they need for their work, and to omit the other details. In our case, domain experts should be able to express the requirements without the technical details. To result in an executable software application, the technical experts should be able to enrich the domain concepts with the additionally needed technical details.

Before describing how our approach finds a remedy to develop software applications, especially Web applications, to solve the described problems, an introduction to Domain-specific Modeling Languages (DSML) based on the Model-driven Software Development (MDSD) paradigm, is given.

3 Using Model-driven DSMLs for Developing Web Applications

A common development approach for Web applications is Model-driven Software Development (MDSD) which provides multiple different levels of abstraction as well as platform-independence. MDSD-based modeling tools, such as model-driven DSMLs, can help multiple stakeholders, with different backgrounds and knowledge, to express relations and behaviors of a domain with familiar notations. The goal is that each stakeholder – maybe with the help of other stakeholders – can easily understand, validate, and even develop parts of solution needed. For instance, domain experts do not have to deal with technological aspects, such as programming APIs or service interface descriptions. Domain experts can assist the technical experts that they can map not well-known domain problems to an appropriate technological model. This leads to an intense collaboration between the different stakeholders and lowers the possibility of misunderstandings [5].

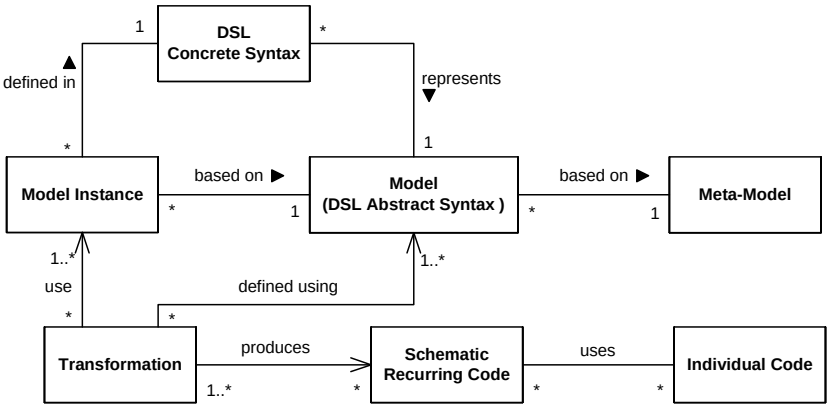

Figure 3. DSLs based on MDSD – relevant artifacts

Figure 3 depicts the major artifacts of MDSD-based DSMLs (see also [3]). An MDSD DSML is based on a meta-model which defines how the domain elements and their relations have to be described [3]. The constructs of the meta-model are used to define models, where a model represents the abstract syntax of the DSML. The abstract syntax defines the elements of the domain and their relationships without considering their notations. Each abstract syntax can be pictured by multiple concrete syntaxes, as described in[3,6]. The concrete syntax describes the representation of the domain elements and their relationships in a suitable form for the stakeholders. Abstract and concrete syntax enable the different stakeholders to define model instances with a familiar notation to represent particular problems of the domain. The ultimate goal of the transformations, which are defined on the model, is to transform the model instances into executable languages, such as programming languages or Web application frameworks. Transformations are used to generate all those parts of the (executable) code which are schematic and recurring, and hence can be automated.

4 Our Approach

To offer expressive and convenient languages for the different stakeholders, our approach provides a horizontal separation of DSMLs into multiple sub-languages, where each sub-language is tailored for the appropriate stakeholders. A suitable separation can be established by splitting the language model into high-level and low-level models, where the low-level models extend the high-level models. Hence, a separation of technical and domain concerns can be established to present only the appropriate concerns to each of the different groups of stakeholders.

In our approach, high-level concerns, relevant for non-technical stakeholders, are distinguished from low-level technical concerns to achieve better understandability for the different stakeholders. That is, high-level DSLs are designed to support the domain experts, enabling them to work in a language which represents the domain concerns with a notation which is close or equal to the domain's terminology. Based on the motivating example in Section 2, high-level domain concerns are *Add Customer*, *Create Customer*, or the definition of the pageflow. In contrast, low-level DSMLs are utilized by technical experts to specify the technical details missing in the high-level DSMLs. These details are needed by the model-driven code generator to transform the model instances, expressed in the DSMLs, into an executable and running Web application. For instance, in the Web application domain, relevant low-level concerns are form, input, submit, variable, or database connection.

Figure 4 outlines our approach of providing tailored DSMLs for all different stakeholders which are involved within the development life-cycle of Web applications. The high-level DSMLs and the low-level ones are based on corresponding language models. Low-level DSMLs provide constructs that are tailored for technical experts, whereas high-level DSMLs provides constructs which are tailored for domain experts, such as managers, project leaders, or project members. The high-level language models tailor the domain's requirements and are extended

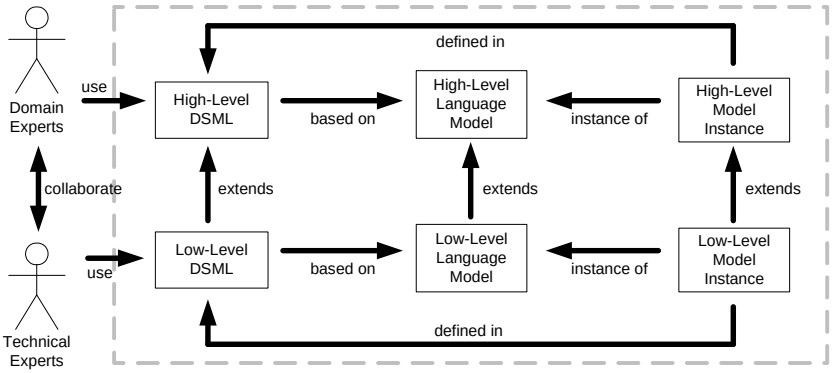

Figure 4. Separation into high-level and low-level

with technical details which are described in the low-level language model. The high-level languages models and the high-level model instances are extended by low-level language models and low-level model instances, respectively.

Following our approach, it is not only possible to provide two different levels of abstractions. Also it is possible to provide multiple different levels of abstractions where each level of abstraction is tailored for the designated stakeholders. The number of different levels of abstractions depends on the problem domain, as well as on the number of the different type of stakeholders.

5 Case Study

This section demonstrates our described approach (see Section 4) based on the motivating example (see Section 2). The following example demonstrates how our approach can be applied for a collaborative definition of the pageflow of Web applications [7]. First, the requirements of the high-level and low-level DSMLs are stated which are based on the requirements of the motivating example and the used existing technology. Then, the language model of the DSMLs is presented. Afterwards the high-level and low-level DSMLs are presented and how they can be used by the appropriate stakeholders. Finally, the co-operation between high-level and low-level DSMLs is described.

5.1 The Requirements of the Web Application

Based on the motivating example of Section 2, the requirements for the domain experts are as follows:

- *Managers* should be able to define the required Web application page.
- *Project Leaders* can define the Web pages of the required Web application.
- *Project Members* are allowed to define the pageflow of the Web pages by using control structures, e.g., `IF`, `SWITCH`, or loops like `WHILE`.

The domain experts should be able to assist the technical experts during the whole modeling process of the Web application. Hence, technical experts need a DSML for mapping the domain concepts to the technical models. The used technology in this example is the JavaServer Faces (JSF) framework [8]. Modeling the pageflow for the JSF framework can be arranged easily because JSF provides an easy and convenient way of defining the pageflow.

After knowing the high-level and low-level requirements of the Web application, the language model can be defined. This is shown in the following section.

5.2 The Language Model of our Approach

To follow our approach of separating the classes of the language model into high-level and low-level ones to provide tailored DSMLs to the appropriate stakeholders is depicted in Figure 5. The implemented language model is based on the requirements of the domain experts as well as on the concepts of the underlying technology, in our case JSF. The separation is depicted by the dotted line, where the upper portion of the figure illustrates the high-level classes and the lower portion the low-level classes. The high-level classes are divided into three views, where each view is tailored for the requirements of the appropriate stakeholders.

First, a description of the classes of the language model is given. Each `WebApplication` consists of multiple `WebPages`. Each `WebPage` consists of a number of `NavigationRules` where each rule points to the subsequent `WebPage` which should be displayed to the visitor of the Web application. Navigation rules can be defined using Java-like `If`, `Switch`, and `While` statements. An `If` statement consists of an `IfBranch` and of an optional `ElseBranch`. A `Switch` consists of multiple `CaseBranches` and an optional `DefaultBranch`. For displaying the same Web pages a certain number of times, `While` loops can be used, e.g., slideshows.

The so far described classes of the model are labeled as the high-level classes because no technical aspects appear within the high-level classes. But, to transform the model instances into an executable Web application, the high-level model must be enriched with technological aspects. This is done by defining classes which are tailored to the JSF Web application framework. In JSF, a `NavigationRule` consist of a number of `NavigationCases`. All navigation rules are assembled to the `JSFConfiguration`. Also, the configuration contains all `ManagedBeans` which store the data of the Web application. However, managed beans are out of scope of this work.

After describing the defined classes and the relationships, each stakeholder's view is described which is based on the above mentioned requirements. Technical experts, in our case JSF experts, can map the defined and required domain concepts, such as the pageflow or the navigation rules, to concrete JSF specific concepts by using the low-level DSML which contains the low-level classes. How domain experts can use the high-level DSML, and how technical experts can use the low-level DSML is demonstrated in the following sections.

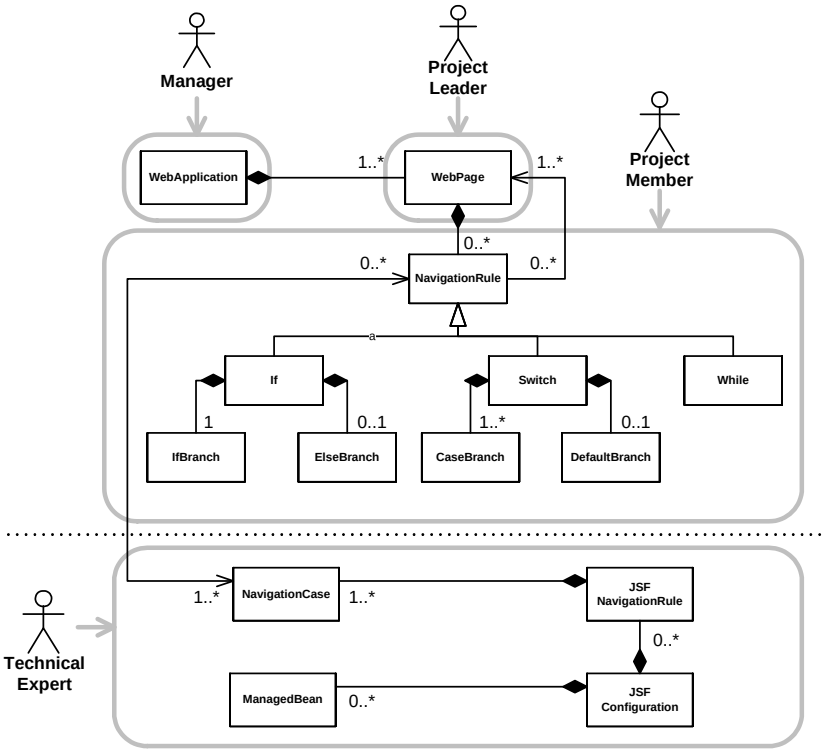

Figure 5. The layers and the stakeholder's views of the model

5.3 The High-level DSML

Traditional design and modeling tools evolved from desktop applications to client-server architectures. As a consequence, an easy way to provide the required stakeholder's views, is to use a graphical web-based high-level DSML. The high-level DSML provides web-based graphical User Interfaces (UI) functionalities, such as Drag-and-Drop, to design pageflows. The benefits of a web-based pageflow design approach are obvious: low requirements to the client, fast access to the pageflow models, and fast propagation of even complex changes in pageflow design.

Figure 6[1] shows a screen-shot of the implemented web-based graphical high-level DSML. After a successful log in into the system, the system knows the role of the user, and the corresponding view can be displayed. Each domain expert – manager, project leader, and project member – has a tailored view which provides only the constructs of the language model which are defined in the above described stakeholder's views. In our case, after the log in of a manager, the manager is only allowed to define a new required Web application. After a manager

[1] The prototype is accessible under: http://www.expressflow.com

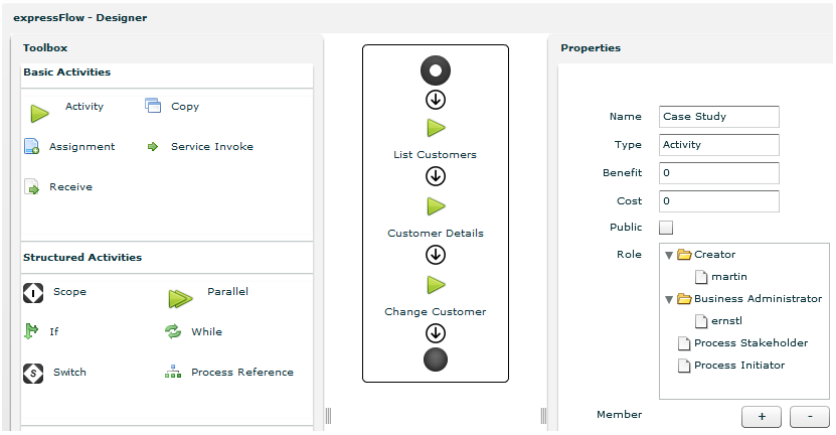

Figure 6. The high-level DSML

defined a new required Web application, the project leaders can define the Web pages of the Web application. Finally, project members can define the pageflow. The role model, to access the pageflow models, is derived from BPEL4People [9]. The manager is refered as the *Creator* of the Web application. This role is determined automatically by the system. The manager adds participants to the Web application as *Business Administrators* to grant full access to the Web application source model. Participants in the role of a *Process Stakeholder* are not allowed to change the Web application source model.

The benefit of this approach is that each stakeholder can change the previous defined Web application's requirements regularly. The outcome of this is a continuos development life-cycle of a Web application which involves all stakeholders and increases their collaboration. How technical experts can use their tailored low-level DSML to map the high-level model instances to the technical concepts is demonstrated in the following section.

5.4 The Low-Level DSML

After each collaborative change by the domain experts, the technical expert can map the high-level model instances to the technical low-level models. Figure 7 illustrates how the technical experts can map and extend the high-level definitions of the modeled Web application to the JSF Web application framework. A demonstration of the mapping of high-level `Switch` statements to JSF navigation rules is demonstrated. The low-level DSML was developed and used within Frag [10].

First, the high-level and low-level language models are imported. Then, the model instance is imported which was defined by the domain experts by using the high-level DSML. The mapping starts with an iteration of all defined Web pages. Within this iteration, an iteration over all defined navigation rules of the

```
## load the high-level model
import HighLevelModel
## load the low-level model
import JSFLowLevelModel
## load the model instance
## the CustomerManagement WebApplication instance
loadWebApp "CustomerManagement"

## iterate over all existing WebPages
foreach webPage [CustomerManagement pages] {
      ## iterate over all existing NavigationRules of the current WebPage
      foreach navCase [$webPage navigationRules] {

            ## check if the current NavigationRule is a Switch
            if {[$navCase isType Switch]} {
                  ## process all CaseBranches
                  foreach cb [$navCase caseBranch] {
                        ## now, do the mapping
                  }
                  ## process the DefaultBranch (if it exists)
                  if {[$navCase defaultBranch]!=null} {
                        ## now, do the mapping
                  }
            }
      }
}
```

Figure 7. Extending the high-level model instance with technical aspects

current processed Web page is done. In our case, this is implement by nested `foreach` loops. If the current navigation rule is a high-level `switch` statement, each case (including the default case) is mapped to a JSF navigation case.

Every time a domain expert changes the requirements of the Web applications by using the high-level DSML, the technical experts have to map the changed requirements to the used technology by using the low-level DSML.

5.5 The Data Exchange between the High- and Low-Level DSMLs

In this work, the high-level and low-level DSMLs are different. The reason for this decision was to provide a graphical syntax for the high-level DSML, making it easier and more understandable to the domain experts. Because technical experts are used to work with textual syntaxes, we decided that the low-level DSML should be based on a textual syntax, making it more usable to the technical experts. The resulting problem is that the modeled requirements of the Web application have to be exchanged between the high-level and low-level DSML. To find a remedy, we decided to provide an XML-based data exchange.

Figure 8 demonstrates our way of enhancing the collaboration between domain and technical experts. After changing any high-level aspects of the Web application within the high-level DSML, the high-level model instances are exported to an XML file which gets imported by the low-level DSML. The technical experts see the changes, map them to the technical model, and extend the high-level model instances with technical details. Afterwards, an XML file

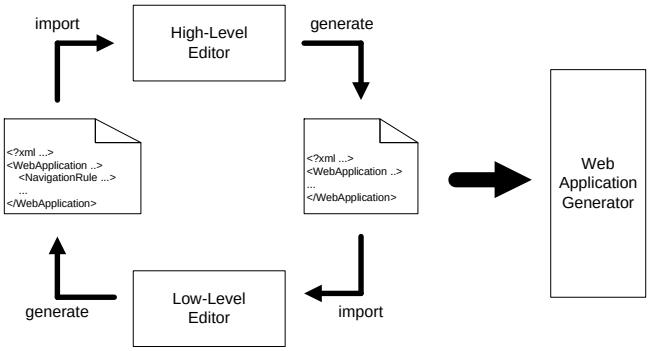

Figure 8. Data exchange between the high-level and low-level DSMLs

is generated, which contains the additionally needed technical aspects to be able to generate a running system. At a certain stage of maturity, the permanently exchanged XML file can be given to the Web application generator which transforms the high-level model instance, which are enriched with technological aspects, to the executable Web application on the desired technology.

Our approach tries to involve all stakeholders into the development process of Web applications, as well as to enhance the collaboration between the stakeholders. During the development life-cycle we discovered some important benefits and drawbacks which are described in the following section.

6 Lessons Learned

This section describes discovered benefits and drawbacks of our approach, which are considerations for future work.

An obvious benefit of the separation into high-level and low-level languages is that domain experts do not have to work with unfamiliar technical details. Also, technical experts do not have to work with not well-known domain concepts. Because of the different expertise of the stakeholders, domain experts have to assist the technical experts that they can understand and map not well-known domain problems to the appropriate technological model. This leads to an intense collaboration between the different stakeholders and lowers the possibility of misunderstandings.

A drawback of our approach can be found in the permanent change of the high-level requirements. For the time being, technical experts have to map the requirements to the technological model every time a change occurs. But, due to the fact that the underlying technology does not change as often as the requirements change, the mapping can be written just once. For example, a high-level `Switch` will always be mapped in the same way to the technological JSF navigation rules. Hence, as future work we have planed to automate the mapping and the transformation to the JSF Web application framework. Because of the close

relation between high-level and low-level language models, we do not know yet how this relation evolves. For future work, we have planned to change the underlying technology and observe how far the high-level model has to be changed. Also, we will observe how changes to the high-level model affect the low-level model.

To find solutions for the drawbacks of our approach, an intense research about related work has to be done. Some of the already found related work is described in the following section.

7 Related Work

Nowadays, many approaches on automated generation of Web applications exists, such as UWE [11], WebML [12], W2000 [13], or OOHDM [14]. To the best of our knowledge, all approaches do not describe how a separation into high-level and low-level is done to provide tailored languages to the appropriate stakeholders. For the time being, our approach is based to the JSF Web application framework. But, to observe the evolution of the relation between the high- and low-level languages if the low-level model gets changed, one of the mentioned approaches can be chosen.

Distante et al. [15] introduce a model-driven approach for combining the development of the Ubiquitous Web Applications (UWA) design framework, the MVC pattern, and JavaServer Faces (JSF). UWA provides conceptual models at a high level of abstraction which can be used by the stakeholders. To provide useful information to the application developers, the UWA conceptual models are transformed to UML-MVC logical models. Both, the conceptual and logical models are platform independent. They are transformed to platform specific models, i.e., JavaServer Faces.

The following approaches are very similar to our approach and we have to discover them in more detail to find solutions for the currently existing drawbacks of our approach. Freudenstein et al. [1] define a methodology for a model-driven Web application development using Domain-specific modeling languages. Their work identifies improvements of the communication and collaboration throughout all stages of the development process of Web applications. Brambilla et al. [16] provide a detailed journey on process modeling for Web applications. Their work extends WebML [12] and WebRatio[2] to map identified process requirements to process modeling for Web applications. Visser [17] introduces WebDSL, a DSL for modeling and developing Web applications based on a rich data model, where a separation into high- and low-level is given too.

8 Conclusion

In this paper we presented an approach to involve all stakeholders – technical and non-technical – in the development process of Web applications. Our approach was exemplified on an introduced motivating example.

[2] http://www.webratio.com, last accessed: April 2009

The approach uses DSMLs which are based on the MDSD paradigm. MDSD is a perfect way to provide multiple levels of abstractions and which can be tailored to the expertise of the appropriate stakeholders. DSMLs are used to provide understandable and reusable possibilities for the stakeholders for modeling the required Web applications.

The high-level DSML, which is tailored for non-technical stakeholders, is based on a high-level language model which does not contain any technological aspects. Hence, non-technical stakeholders, also called domain experts, can work with a language that is tailored to their expertise and do not have to handle with non-understandable technical details.

For an automated generation of an executable Web application, the high-level language model has to be enriched with additionally needed technical aspects. This is solved by the low-level language model which is based on the used technology. Technical experts have to map to required domain concepts on the low-level language model within a DSML which provides constructs that are familiar to the technical expert.

To model the requirements of Web applications, the domain experts assist the technical experts during the mapping of the domain concepts to the technological model. Hence, technical experts do not have to handle with well-known domain concepts. The assistance leads to an intense collaboration between all technical and non-technical stakeholders which are involved within the whole development life-cycle of a Web application.

Acknowledgement:
This work was supported by the European Union FP7 project COMPAS, grant no. 215175.

References

1. Freudenstein, P., Buck, J., Nussbaumer, M., Gaedke, M.: Model-driven Construction of Workflow-based Web Applications with Domain-specific Languages. In: MDWE. (2007)
2. McDonald, A., Welland, R.: Agile Web Engineering (AWE) Process: Multidisciplinary Stakeholders and Team Communication. In Lovelle, J.M.C., Rodríguez, B.M.G., Aguilar, L.J., Gayo, J.E.L., del Puerto Paule Ruíz, M., eds.: ICWE. Volume 2722 of Lecture Notes in Computer Science., Springer (2003) 515–518
3. Thomas, S., Markus, V., Krzysztof, C.: Model-Driven Software Development: Technology, Engineering, Management. John Wiley & Sons (2006)
4. Oberortner, E., Zdun, U., Dustdar, S.: Domain-Specific Languages for Service-Oriented Architectures: An Explorative Study. In Mähönen, P., Pohl, K., Priol, T., eds.: ServiceWave. Volume 5377 of Lecture Notes in Computer Science., Springer (2008) 159–170
5. Douglas C. Schmidt: Model-Driven Engineering. IEEE Computer **39**(2) (February 2006)
6. Baar, T.: Correctly defined concrete syntax. Software and System Modeling **7**(4) (2008) 383–398

7. Oberortner, E., Vasko, M., Dustdar, S.: Towards Modeling Role-Based Pageflow Definitions within Web Applications. In: Proc. of the 4th International Workshop on Model-Driven Web Engineering (MDWE 2008). Volume 389 of CEUR Workshop Proceedings., Toulouse, France, CEUR-WS.org (September 2008) 1–15
8. Sun Developer Network: JavaServer Faces Technology
Available online at http://java.sun.com/javaee/javaserverfaces/.
9. IBM, SAP: WS-BPEL Extension for People http://www.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/.
10. Zdun, U.: The Frag Language http://frag.sourceforge.net/.
11. Andreas Kraus and Alexander Knapp and Nora Koch: Model-Driven Generation of Web Applications in UWE. In: MDWE. (2007)
12. Stefano Ceri and Piero Fraternali and Aldo Bongio: Web Modeling Language (WebML): a modeling language for designing Web sites. Comput. Netw. **33**(1-6) (2000) 137–157
13. F. Garzotto, L. Baresi, and M. Maritati: W2000 as a MOF Metamodel. (2002) In The 6th World Multiconf. on Systemics, Cybernetics and Informatics-Web Engineering track.
14. Daniel Schwabe and Gustavo Rossi: An Object Oriented Approach to Web-Based Application Design. Theor. Pract. Object Syst. **4**(4) (1998) 207–225
15. Damiano Distante and Paola Pedone and Gustavo Rossi and Gerardo Canfora: Model-Driven Development of Web Applications with UWA, MVC and JavaServer Faces. In Baresi, L., Fraternali, P., Houben, G.J., eds.: ICWE. Volume 4607 of Lecture Notes in Computer Science., Springer (2007) 457–472
16. Brambilla, M., Ceri, S., Fraternali, P., Manolescu, I.: Process modeling in web applications. ACM Trans. Softw. Eng. Methodol. **15**(4) (2006) 360–409
17. Visser, E.: WebDSL: A Case Study in Domain-Specific Language Engineering. In Lammel, R., Saraiva, J., Visser, J., eds.: Generative and Transformational Techniques in Software Engineering (GTTSE 2007). Lecture Notes in Computer Science, Springer (2008)