TOWARD REUSABILITY IN WEB MODELING
Using QVT Relations

Ali Fatolahi, Stéphane S. Somé and Timothy C. Lethbridge
School of Information Technology and Engineeing, University of Ottawa, 800 King Edward, K1N 6N5, Ottawa, Canada

Keywords: MDD, QVT, Web, Reusability, Meta-model.

Abstract: In this paper, a model-driven approach for web development is presented. The approach contains two important elements that serve reusability: an abstract model and a set of transformations. Transformations act as the chaining feature of model-driven development (MDD); that is transformations add to the value of models by transforming them to those of the desired type. As a standard for developing transformations, QVT relations are used in this paper to specify mappings from a high-level model to an abstract model of web-based applications. This model is abstract since it does not rely on any specific web platform but on the common features of web applications. Having this model and its corresponding transformations, model-driven web development for specific platforms becomes faster and more reusable.

SCIENCE AND TECHNOLOGY PUBLICATIONS

1 INTRODUCTION

Models form the value of model-driven development (MDD) methods. Transformations add to the value of MDD by processing raw models and by creating more detailed ones that could be used for the development of actual applications. We aim at maximizing the level of value re-use as well as minimizing the length of transformations that create value or add to it. Thus, we present an abstract model of web applications that is reusable for several web platforms as well as a set of QVT relations to transform requirements to the abstract web models.

We have added an intermediate level termed Abstract PSM (APSM) to the three conventional levels of MDA, i.e. CIM, PIM and PSM. The APSM, an intermediate between the PIM and PSM in the context of Web applications, is concerned with models specified with respect to common features of different web platforms. The APSM is platform-specific in the sense that it describes features specific to the abstract web platform; it is also abstract since it does not contain details of specific web platforms but only their shared features. Two sets of transformations are needed from PIM to PSM: one to map PIM to APSM and the second one to map APSM to PSM.

An important aspect of our work is the usage of QVT relations as a means of formalizing model-driven transformations. In this paper, we present an approach based on QVT relations to be used for the generation of APSM models. Hence, the PIM-to-APSM part of the transformations could be re-used for all applications. Only the QVT relations required for generating a specific PSM from the APSM need to be supplied. Since the APSM is semantically closer to PSMs, this latter part is easier to develop in comparison to the conventional PIM-to-PSM transformations.

The importance of web applications as well as the ever-increasing volatility of web technologies postulates the need to benefit from models that have been created for previous projects. In order to achieve this, one needs either web models that are adaptable with virtually any platform or automated transformations that enable transforming models of different platforms to each other. In this paper, we use a combination of the two.

The APSM, which is used as the abstract web model in this paper, is a reusable model because it can be used to describe models of web applications regardless of their platforms. Hence for example, the APSM of a web application once deployed using a .Net-based technology could be reused to create the same application using a GWT-based framework as well.

Also, the transformations that map the APSM to specific platforms come in handy here. These facilitate model reuse by relieving the developer

Fatolahi A., S. Somé S. and C. Lethbridge T..
TOWARD REUSABILITY IN WEB MODELING - Using QVT Relations.
DOI: 10.5220/0003290703830386
In *Proceedings of the 7th International Conference on Web Information Systems and Technologies* (WEBIST-2011), pages 383-386
ISBN: 978-989-8425-51-5
Copyright © 2011 SCITEPRESS (Science and Technology Publications, Lda.)

from low-level tasks required to create the platform-specific designs. At the ultimate case, with a completely automated process, the task of reproducing the same application for different platforms using different technologies becomes a matter of changing the automated code generation module that targets the new platform.

2 META-MODEL FOR ABSTRACT WEB

According to Figure 1, an application has several use cases, a number of these use cases may be startup use cases. User interfaces may be defined for every application. For web-based applications, it is necessary to recognize different views for different users; this is realized using the association of actors and user interfaces. States may be associated with presentations; this results in a presentation state. A presentation is a special UI Composite, which means it could contain other UI Components. UI Composites are translated to pages, forms, tables or panels depending on the specific platform the APSM will be mapped to. UI Components may be associated to each other through a *FieldOperation*. This allows client-side operations to be defined. A UI component is associated with a data composite in order to model the data support required. Data composites are composed of data entities. A data composite can also participate in an association with another data composite, where one data composite is used as the basis of selecting data from another one. For example, when selecting a country affects the list of available provinces such association would be created.

Every data composite is supplied with a service class, which manages the flow of information to and from the data composite. An attribute *crudNature* is added to the class *Operation* in order to determine the type of service operation and is used for code generation. Also, in order to model the call structure from controller operations to service ones, an association is added from the class *Operation* to itself. A special UI composite, *OperationTrigger*, is used to represent the information submitted to the server side, for example, as part of a web form.

Transitions may carry events. Events can be either signal or call events. Signal events represent the events fired by UI Components such as Operation Triggers, i.e. submit buttons. Call events are rather called by controllers to handle the logic of the application. These, often, call a data service.

In order to be able to move between platforms, a web model must be able to specify Web 2.0 applications as well. The model is capable of supporting Web 2.0 applications at an abstract level. A presentation state may be composed of several presentation units, which allows independent update of different sub-units within the same presentation unit. Also content-oriented UI elements may accept feedback, where the feedback is itself a content-oriented UI element. This enables an interactive content as required by Web 2.0. Also, by having content-oriented UI elements attached to items found for every search event, the whole content becomes searchable.

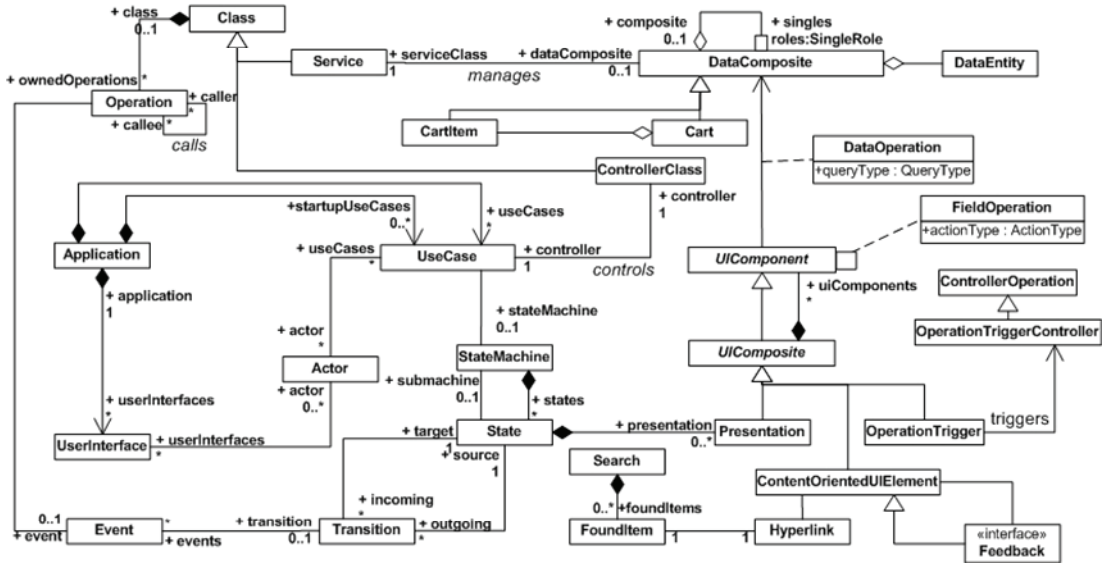
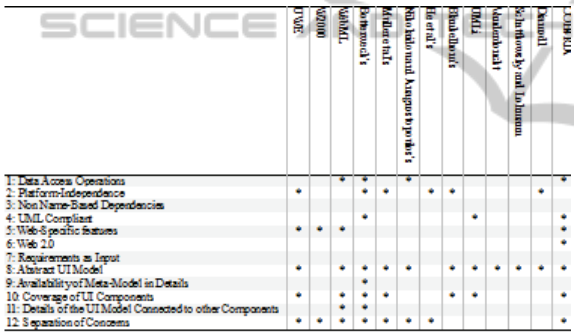


Figure 1: A Meta-model for Abstract Web Applications.

In order to achieve reusability using model-driven transformations, it is critical to have transformations that could be traced forward and backward through elements that map platforms to each other. According to Cichetti and Di Ruscio (2008), most of the existing web models do not adequately support this because they lose track of changes by relying on name-based associations. Our meta-model avoids name-based associations by creating concrete associations such as the one between *UIComponent* and *DataComposite*, which provides the same type of support as suggested by Cichetti and Di Ruscio (2008) in terms of an association from *DataCompositionWLink* and *CompositionWElement*.

Despite the popularity of the existing web models, as shown in Table 1, none of the existing models meets all the requirements of a reusable model in the context of a model-driven approach.

Table 1: Comparative Study of the Existing Web Models.



3 AN EXAMPLE

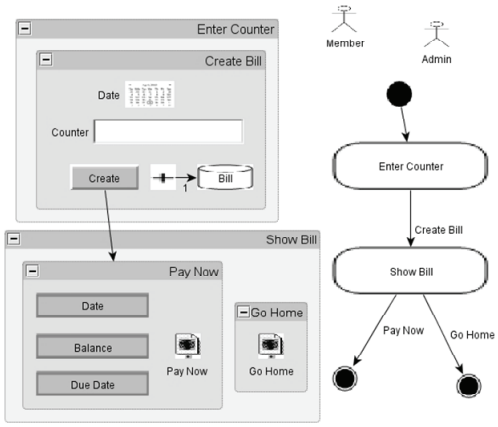


Figure 2: A Sample Input Model.

Figure 2 shows a sample input model. The figure shows that two actors, *Member* and *Admin* may access this process. This is related to a billing use case. The User enters a counter number and requests a bill. The system calculates and shows a bill providing an option of instant payment or to return. The group boxes at the left show the corresponding presentations of every state. There could be more than one presentation unit per page. The presentation model of the state, *Enter Counter* is only composed of one entry form that adds a new entry to the data unit *Bill*. The state, *Show Bill* is related to a page composed of two units. First unit shows the information of the recent bill based on the created bill in previous step. The second unit is simply a link to home page.

The state *Enter Counter* is translated to a presentation state, which leads to another state through signal event *Create Bill* with two parameters *Date* and *Counter*. The state *runState1* is only inserted to enable the activation of the call event on the outgoing transition that calls the operation required to create the bill. In case this is successful, the object *theBill*, which carries the newly created bill, is sent along the transition that ends at the final state *Online Payment*.

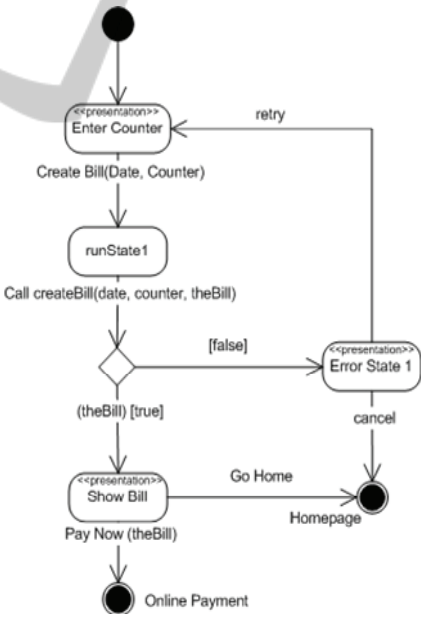


Figure 3: The APSM Mapping of figure 2.

Figure 4 presents the classes required to support the input model. The data composite *BillComposite* is composed of only one class, *BillEntity*. The attributes of *BillEntity* are selected so that there is one attribute per UI component in Figure 3. The class *BillService* handles the CRUD operations. The controller *BillingController* needs an operation to

handle the creation of a bill. This operation is basically required to create an empty *BillComposite* object, to call the corresponding service and to transfer this object to the next state as required for payment.

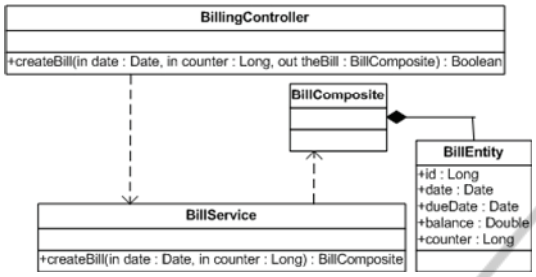


Figure 4: The APSM Classes of *Billing*.

4 MAPPINGS

Figure 5 presents a summary of the information used from the PIM as well as those created at the APSM level. According to Figure 8:

- Data objects and navigations through data objects are created based on data associations found within the UI model.
- The contents and the structure of web pages are determined by the contents of the presentation states and transition flows.
- Transitions and states from the input model are also used to create events and operations used for the generation of the behaviour and controllers of the application.
- The generated behaviour and controllers are used in turn to build the data access services in combination with data associations from presentation states. It is also used to map parameters to attributes within the data model.

The following transformations were implemented:

1- PIM-to-APSM: This set maps the input model such as the one in Figures 2 to the APSM model such as the one of Figures 3-4.

2- APSM-to-AndroMDA: This mapping creates a fully detailed AndroMDA model, which may be transformed to code using AndroMDA .

3- APSM-to-WebML: The target in this mapping is a specific configuration of the WebML-based tool, WebRatio.

4- APSM-to-GWT: maps the APSM to a platform, based on Google Web Toolkit. This allows us to evaluate the Web 2.0 modeling capabilities of the abstract meta-model.

Although it is possible to use the approach without the PIM-APSM transformations, this set helps bringing reusability to an even more abstract level that is reuse at the level of requirements model.

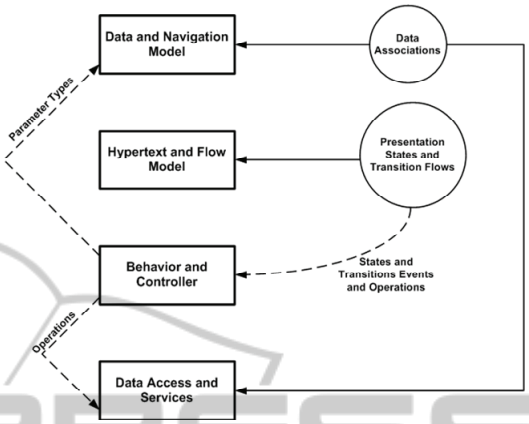


Figure 5: Summary of the transformations.

5 IMPLEMENTATION

The implemented approach is named MODEWIS, which stands for MOdel-driven DEvelopment of Web Information Systems. Figure 6 shows how the MODEWIS interacts with other components in this implementation. Several case-studies, tools and transformations are developed in the context of MODEWIS (2010).

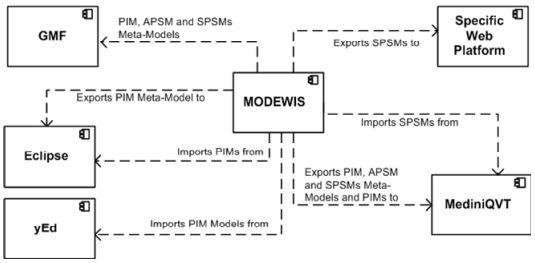


Figure 6: The Architecture of MODEWIS.

REFERENCES

AndroMDA, www.andromda.org, 15-02-2007.

Cicchetti, A. Di Ruscio, D. Decoupling Web Application Concerns through Weaving Operations. In: Science of Computer Program-ming 70(1) 2008. pp. 62-86.

Google Web Toolkit, Google Code, code.google.com/webtoolkit/, June 2010.

MODEWIS, modewis.blogspot.com/, 12-1-2010.

WebML, www.webml.org, May 5, 2008.

WebRatio, www.webratio.com, 6-5-2008.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS