

A Meta-Model for Model-Driven Web Development

Ali Fatolahi, Stéphane S. Somé, and Timothy C. Lethbridge

(School of Electrical Engineering and Computer Science, University of Ottawa, Canada)

Abstract Several model-driven development (MDD) techniques for web applications exist; these techniques use meta-models for defining transformations and designing models. In this paper, we propose a meta-model for abstract web applications that can be mapped to multiple platforms. We extend a UML-based model to support specific features of the Web and Web 2.0 as well as to establish a bridge to functional and usability requirements through use cases and user interface (UI) prototypes. The meta-model also helps avoid a common MDD-related problem caused by name-based dependencies. Finally, mappings to a number of specific web platforms are presented in order to validate the appropriateness of the meta-model as an abstract web model.

Key words: web; meta-model; MDD; UML

Fatolahi A, Somé SS, Lethbridge TC. A meta-model for model-driven web development. *Int J Software Informatics*, Vol.6, No.2 (2012): 125–162. <http://www.ijsi.org/1673-7288/6/i117.htm>

1 Introduction

Model-driven techniques have been used in the area of web development to a great extent; examples are found in Refs. [1-10]. Such methods require precise definition of the meta-models used as the source and the target of transformations. In order to formally define those transformations, the meta-model postulates the involvement of unambiguous links between different modeling elements and their mappings. Additionally, a meta-model for web applications should be defined to cope with the World Wide Web platform volatility. In this paper, we present a UML-based meta-model of web applications to be used in the context of model-driven methods targeting several specific platforms.

Many of the model-driven web engineering approaches are tuned toward specific platforms or develop only certain parts of web applications. These approaches generally use meta-models adapted to their targeted platforms. However, approaches that can flexibly target several platforms are needed because of the rapid evolution of Web technologies, as well as requirements to deploy applications on multiple platforms such as desktop and mobile-based platforms. The ability of a model-driven method to target multiple platforms depends on the abstractness of the meta-model used. Such a meta-model must allow the description of relevant features of web applications independently from the specificities of specific platforms. Additionally, transformations mapping from abstract to specific web descriptions must be expressible in

Corresponding author: Ali Fatolahi, Email: ali.fatolahi@gmail.com

Received 2010-12-29; Revised 2011-06-06; Accepted 2011-09-26; Published online 2012-02-16.

a flexible way. In this paper, we propose such an abstract meta-model. Mappings that transform abstract models to specific platforms defined based on AndroMDA^[8], WebRatio^[9] and Google Web Toolkit (GWT)^[11] are also presented for validation.

Our meta-model is an abstract model for web information systems. Models of web applications that are built upon this meta-model belong to the Platform-Specific Models (PSM) level of MDA^[12]; we name this type of PSM, Abstract PSM (APSM). The APSM is defined upon an abstract model for web-based applications. It encompasses a set of modeling elements delimiting universal necessities of web-based applications regardless of the platform on which the application is deployed. We distinguish the APSMs from Specific PSMs (SPSMs) that pertain to specific web implementation platforms such as. Net or J2EE. 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.

Our meta-model is typically used for specifying transformations from MDA Platform-Independent Models (PIMs) to APSMs. These PIM-to-APSM transformations are completely re-usable when for instance a web application needs to be ported to different specific platforms. Only the transformations required for generating a specific PSM from the APSM need to be supplied. Since an APSM is semantically closer to a SPSM, this latter part requires less effort to develop in comparison to the conventional PIM-to-PSM transformations. A smaller subset of our meta-model is used for specifying the PIM.

The meta-model has been designed following four principles. First, in order to ensure MDD transformations can run unambiguously, semantic associations between elements of different viewpoints are supplied as required. Second, the APSM is connected to functional and usability requirements through state machines that model use cases, as well as UI prototypes. This allows modeling of the input models at an abstract level. Third, the APSM integrates specific features required for Web 2.0 applications. Finally, our meta-model extends UML. UML was selected as it is an established modeling standard. We refer to the version 2.1.1 of UML^[13] in this work.

In this paper, an abstract platform^[100] refers to the abstract web as a model combining common features of web technologies and platforms. The APSM is hence a model pertaining to the abstract web platform. A specific platform is a collection of concrete technologies, guidelines and configurations that are used to develop executable web applications. The SPSMs are hence defined upon specific platforms. The gap between APSM and every SPSM is subject to adding the details of the specific platform, on which the SPSM is dependent, to the APSM. We use the term “platform” generically. Hence, in this paper, a PIM is platform-independent in general while a PSM is platform-specific in general. Therefore, The APSM and the SPSMs are both PSMs generically.

The rest of this paper is organized as follows. In Section 2, we will enlist the requirements of designing an abstract web model. In Section 3, we review the existing models of web applications and will discuss the features that are not properly addressed so far. In Section 4, we will present our meta-model and in Section 5 some examples are introduced. In Section 6, the meta-model is evaluated by different mappings to specific platforms. In Section 7, we will discuss the implementation, results

and usability issues. Finally, Section 8 concludes this paper.

2 Requirements of an Abstract Web Model

An abstract web model should allow modeling of common properties and features present in concrete web models as well as web engineering coding frameworks. These requirements could be elicited from existing web models as well as more recent web technologies. More specifically, we are interested in requirements that facilitate the usage of MDD techniques. We aim at designing a model with the most common features as well as the most required ones and we will use our automated transformations to generate specific features of specific platforms.

We may start with the fact that most web application frameworks are based on the Model-View-Controller (MVC) or a similar pattern for the separation of concerns. ASP.NET^[86], ColdFusion on Wheels^[87], Apache Struts^[88], Java Server Faces (JSF)^[89], JBOSS Seams^[90], Spring, Ruby on Rails^[91], Aida/Web^[92] and Catalyst^[93] are only some examples. The most recent version of .Net framework has moved further on by providing default controllers and stipulating strict packaging rules for separation of concerns in accordance with MVC 2.0. The notion of “controller” is central in this structuring scheme. Even a framework such as GWT, which does not explicitly support MVC, supplies entry points and an asynchronous event-management framework for controlling the application. It is therefore inferred that a web model needs a controlling feature composed of a controller class and its supporting operations. The controller acts as a central decision point for the application or its subsequent features. The implementation of this controller may be according to the original MVC pattern^[83], variations of MVC such as Ref. [69] or non-MVC patterns such as the ones of service-oriented applications; this is left to the target specific platforms.

Web application frameworks all support data access. Some, such as Ruby on Rails provide even more facilities for database migration. A variety of methods are used by different web application frameworks for data access, interchange, format conversion and presentation. It is therefore, necessary to model abstract data handling features that can be mapped to various platform-specific routines. The data supplied through the data interfaces of the existing frameworks is often required to be serialized to be usable at the logical layers of the application. Hence, there is a need for two types of data containers, one to be used at the data access layer and a second one to be transmitted in between layers.

Most of the existing web application frameworks share security features such as encoded sessions or authentication mechanisms. An abstract web model needs elements representing secure pages, operations and services as well as login/logout mechanisms and session/page variables. Some other frameworks such as CppCMS^[84], Fusebox^[85] and GWT do not supply such features as built-in utilities, yet these frameworks can also be used for implementing security features as well. For example, GWT is widely based on Ajax and Google services that require and enable secure connections.

Web application frameworks support the creation of user interfaces (UIs) using different types of UI components. Many of these frameworks rely on their host language such as Java, Perl or PHP for the definition and deployment of UI components. Some such as GWT and ASP.Net provide custom types. Nevertheless, UI components

trigger events and use data. As a result, an abstract web model should supply different abstract UI components and support processes that relate them to operations required to furnish with data and to trigger events.

Web 2.0 has added several new features to web development. Several web application frameworks such as GWT, ColdFusion on Wheels and Apache Struts integrate support for Ajax, which is now seen as the most popular Web 2.0 facilitator. Other Web 2.0 technologies such as Flash and HTML 5.0 are also popular, especially with regard to mass data processing and multi-media applications such as media sharing sites and mobile applications. Web 2.0 requires that separate parts of a web page be updatable independently, the contents be searchable and the user be able to interact with the contents in real-time.

Since our approach is MDD-based, it is important to consider requirements from these techniques as well. As described by Cicchetti and Di Ruscio^[43], most of the existing models are affected by name-based mapping rules that can cause ambiguities with regard to the separation of concerns. Semantic relationships instead of name-based associations should be used in meta-models. MDD techniques often establish a mapping between elements of different viewpoints based on their names. This is a useful technique but one that makes it difficult to back-track and to identify the origin of mappings because of changes applied by developers to element names as well as the one-to-many nature of most of the mappings. Another MDD-related requirement is the selection of an appropriate language for the meta-model. Many domain-specific languages (DSLs) for web development exist but another option would be a UML-based language. Using UML, one can benefit from the popularity of an established standard as well as its integration to different tools and methods.

Finally, from the viewpoint of a software engineering technique, we take notice of two important features. First, it is important that lower-level models be created in accordance with higher-level requirements^[70–72]. Although there is a limited capacity for directly addressing this aspect in an abstract web model, different researchers and practitioners^[73–80] have observed that the more a web modelling approach include the possibility to express requirements-related features, the more beneficial it becomes. It also has been observed that a barrier that prevents developers from using MDD techniques is their reluctance to learn a new complicated technique after becoming experts in one technology^[81]. To alleviate this barrier, it is important that a web application can be defined using a simplified input language, and to provide automated transformations from that language.

In summary, the following are main requirements for an abstract web model used in MDD:

- 1) Support for data access and event handling. This includes controlling mechanisms and support for data services.
- 2) Platform-Independence. This ensures the model is abstract and can be mapped to several specific web platforms.
- 3) Support for MDD necessities. This includes the fact that modeling elements, when required, must be connected using semantic associations. This category also addresses the ease-of-use by offering a simplified subset of the meta-model as an input language along with the required transformations to map the input to the complete model.

- 4) UML support. This is seen as a requirement for defining the meta-model to enhance the popularity of the meta-model and its portability with respect to other tools, models and techniques.
- 5) Support for Security.
- 6) Support for Web 2.0.
- 7) Support for expressing requirements. This includes functional requirements as well as UI components that represent the expected UI prototype of the application.

3 Models of Web Applications

The most related models are those that address web modeling along with UI modeling. WebML is introduced as a modeling language for designing web based applications by Ceri *et al.*^[14]. WebML provides a notation to define the UI presentation layer and the semantics to specify web-based applications in a multi-layer approach. According to WebML, a web-based application is defined in terms of: Data modeling, Hypertext modeling, Personalization to give different users different viewpoints, Presentation to add the look-and-feel, and Integration of business processes and Web services.

Botterweck^[17] presents an abstract model for specifying web-based applications supported by a UI model and a data model along with connections to the UML state machines. Botterweck's model aims at specifying multiple UIs, which is close to our objective. It extends UML in order to model an application in different layers. Modeling elements are provided for state machines, presentation layer, data layer and services. In our work, we have adapted Botterweck's model as the basis of our model. Other models such as WebML provide the same support we need but Botterweck's has the advantage of being UML-based.

Some web engineering processes have also suggested web models as parts of their framework. As an example, UWE^[15] bears its own abstract model of web-based applications as well as an abstract UI model. UWE was broached as an extension to the Rational Unified Process (RUP)^[16] for web development. UWE offers means of defining requirements and modeling navigational aspects as well as presentation features of a web application. These, along with the contents and the required processes are used to generate the application using the transformations. It is, however, noteworthy that as its origin – RUP – UWE acts as a framework rather than a concrete method. The most proper usage of UWE, therefore, is perhaps to define model-driven methods on top.

Muller *et al.*^[18] provide another abstract model with the same intentions as Botterweck's but based on a different approach. Muller *et al.* break up the PSM level into two levels: One is named platform-dependent PSM and the other is technology-dependent PSM. Since the model is provided for web in general, the term 'platform' is not applicable to web itself but to specific platforms and technologies. Such a separation results in transformation from PIM to platform-dependent PSM relatively longer than the ones from platform-dependent PSM to technology-dependent PSM. A difficulty with Muller *et al.*'s approach is thus to identify and differentiate the terms platform and technology.

There are abstract models that address web applications but do not specifically describe the details of UI modeling. Nikolaidou and Anagnostopoulos^[5] suggest a

common UML meta-model for web-based information systems to tackle performance problems in the functional and physical layers. W2000 is a meta-model for web applications by Baresi et al.^[19], which provides a multi-layer architecture for web-based applications in accordance with the Model-View-Controller (MVC) pattern^[20]. He et al.^[21] report an interesting approach to define a role-based abstract model for web-based applications. However, the method is instead a general method for creating abstract web models. Another effort to support web modeling using MVC is presented by Lowe et al.^[97] as an extension to UML. Based on the motivations found in Ref. [98], Lowe et al. address the functional aspects of a web application and create a conceptual model that relates different layers of the web application and hence alleviate following business needs through layers of applications.

Another group of the existing models are those that address abstract UI models with less connection to web applications. Blankelhorn^[22] has provided a UML profile for GUI layouts. Da Silva and Paton^[23] describe a how-to for designing user interfaces using UMLi that is an extension to UML and compares the results with the ones from the UML itself. Vanderdonckt^[24] presents an MDA-compliant environment for designing user interface models of information systems in general. Schattkowsky and Lohmann^[25] introduce a general method for designing platform-independent UIs but the method is not supported by any specific meta-model. Diamodl^[26] is a modeling language to describe the data-related logic of UIs. This includes the links, data flows and data collection gates. Diamodl is very limited and cannot scale up to cover higher-level aspects of UI modeling because it only models the data interchange between abstract atomic data units at the programming level.

We may also mention other efforts devoted to synthesize UI models in different ways. Li et al.^[27] illustrate a how-to UML-based approach to devise the architecture of web-based applications in a three-tier framework. Kavaldjian^[4] and Bogdan et al.^[28] describe a method to extract a UI model from a discourse model in a general sense. Costa et al.^[29] present an MDA-compliant method to devise a step in user interface design for web applications in accordance with UML version of ConcurTask-Tree (CTT). The new language is called *Canonical Abstract Prototype* (CAP) that is an abstract language to define user interfaces. The goal of the article is to transform the UID (user interface definition) model to a UML-compliant specific model for interoperability. The PIM is use case-driven. Conceptual Architecture, Presentation Model and Dialog Model are elements of both PIM and PSM. The authors suggest an example mapping from CAP to HTML as a specific platform. The transformation from PIM to PSM and to code is executed using the PHP application Model2Code. De Souza et al.^[30] have published another model-driven approach to generate web UIs. The method covers the layers of MDA as follows: CIM with use cases, PIM with analysis and presentation models, PSM with design, navigation and UI design model. The method starts with use cases and spans through analysis, presentation, design and navigation models. The final result is a UI design model, which is followed by the implementation of the UI. An implemented Eclipse plug-in hard-codes transformation rules required to generate JSF code. Sukaviriya et al.^[31] introduce a process framework for model-driven UI design based on iterative/incremental approaches. Arraes, Nunes and Shwabe^[32] present a combined approach of model-driven development and domain specific language to derive a rapid software prototyping for

web applications. A tool named HyperDe takes conceptual instances, a navigation model and an abstract UI to produce HTML pages.

Approaches that address parts of the web UI form another group of the related work. As an example, Freudenstein et al.^[33] present a modeling language to specify web dialogs, which includes a domain specific language (DSL), based on Petri-nets; a Domain-Interaction Model, which is a simple interface to connect the models to DSL and work with the application that provides the model; Solution Building Block that is a software component to execute the dialog model. The modeling notation is capable of defining data elements and their interactions as well as the user interface components used within the dialogs. Tongrungrrojana and Lowe^[34] use WebML in order to present the language WEID at a higher-level of abstraction to address the inefficiency of the existing web/information modeling languages when dealing with information exchange models at the level of business processes. This is a companion to Tongrungrrojana and Lowe's previous work about WebML+^[35], a language for modeling information exchange in the workflow models. Some older approaches such as LHM introduced by Wan et al.^[37] and GHMI by Wan and Bieber^[36] are early efforts to integrate classic information systems with hypertext features. Another similar approach is taken by Whitehead et al.^[10] to bring repositories to the hypertext level so that they could be visualized using the web.

Finally, Web 2.0^[38] has been also addressed in the model-driven community. Specific model-driven approaches targeting Web 2.0 exist. For example, Valverde and Pastor^[39] describe a meta-model to specify technical details of a Web 2.0 platform based on OOWS^[40]. These include the features related to Rich Internet Applications (RIA) such as UI widgets as well as the event handling mechanism. Hernández et al.^[3], in another effort, explain an approach toward defining a meta-model for the definition of the conceptual aspects of web 2.0 as a venue for social networking. The authors elaborate a model-driven overture to social networking that includes modeling elements such as actors, objects and activities and their relationship.

Table 1 lists some important factors to compare the existing models using our requirements. Following is a description of each row of Table 1:

- **Data Access Operations**, this indicates if the meta-model supports operations for data handling at an abstract level.
- **Platform-Independence**, this refers to the fact that a model is abstract and also if the model has been validated against at least two different specific platforms.
- **Non Name-based Dependencies**, this means the model does not depend on the name-based analysis rules to interrelate elements of different viewpoints.
- **UML-Compliant**, which refers to the fact that the model extends UML.
- **Web-Specific Features**, this means the model supports web-specific features in opposition to abstract-web features.
- **Web 2.0**, this verifies if the model explicitly supports Web 2.0. For example, in WebML, one may use script units for inserting Ajax code to support some Web 2.0 capabilities but this is only a work-around. We rather look into a model that allows every element to be searchable, to accept feedback and to be updatable independently.
- **Requirements as Input**, this row indicates if the model integrates requirements with lower-level models.
- **Abstract UI Model**, this suggests that the model presents the elements

required to model an abstract UI model.

- **Availability of Details**, this verifies if the details of the meta-model are available. With regard to MDD techniques, such details are necessary for extending the model as well as creating the transformations that map the model to other models, languages and tools.

- **Coverage of UI Components**, by this, we point to the fact that a meta-model has elements for modeling specific UI components such as tables, buttons and input controls.

- **Integration of UI Components and the Meta-Model**, which means if the supported UI components are integrated with behavioral and navigation aspects of the meta-model using meaningful associations.

- **Separation of Concerns** that is if the model addresses different viewpoints and suggests their proper integration.

As Table 1 shows, Botterweck's model scores higher in terms of support of the above elements. It is worth noting that Botterweck's model is a general UI model not restricted to web applications. An advantage of this model compared to web modeling languages such as WebML is that it is based on the UML, recent efforts have however focused on complying WebML to the UML^[44]. It is also worth mentioning that Botterweck's model provides support for advanced UI elements such as multimedia contents unlike most other models. Table 1 also draws attention to the fact that none of the studied models are grounded based on requirements as an input along with models of UI, data and expected operations that satisfy those requirements. We selected Botterweck's model as basis for our model but, as indicated in Table 1, Botterweck's model needs to be extended in order to address items 3, 5, 6 and 7. More details are found in Ref. [48].

Table 1 Comparative study of existing abstract models

	UWE	W2000	WebML	Botterweck's	Muller et al's	Nikolaïdou and Anagnostopoulos's	He et al's	Blankholm's	UMLi	Vanderdonckt	Scharkowsky and Lohmann	Diamodl	OO4RIA
1: Data Access Operations			*	*		*							*
2: Platform-Independence	*			*	*		*	*				*	
3: Non Name-Based Dependencies				*									
4: UML Compliant				*					*				*
5: Web-Specific features	*	*	*										*
6: Web 2.0													*
7: Requirements as Input													
8: Abstract UI Model	*		*	*	*	*		*	*	*	*	*	*
9: Availability of Meta-Model in Details				*									
10: Coverage of UI Components	*		*	*	*			*	*				*
11: Details of the UI Model Connected to other Components			*	*									
12: Separation of Concerns	*	*	*	*	*	*	*						*

Despite the plurality of related work, there is still a need for a new abstract web

model for model-driven web engineering for the following reasons.

1. The existing models including the ones addressing data-centric or information-based web applications - such as the ones suggested by Ceri *et al.*^[1], Nikolaidou and Anagnostopoulos^[5] and Baresi *et al.*^[19] do not tend to cover the data access layer; rather they focus on the presentation and navigation required to present this data and information at the hypermedia level; WebML is an exception.

2. Most of the existing models are usually defined toward a PSM. Exceptions such as UWA^[41] and Sakowicz *et al.*^[42] end up with models that are very general and too abstract. As a result, most of the research leads to approaches that are either hard to map to specific platforms or hard to adapt with other specific platforms.

3. As described by Cicchetti and Di Ruscio^[43], most of the existing models are affected by the name-based mapping rules that prevent effective model-driven transformations be developed.

4. Traditional web modeling languages including WebML are not based on UML/MOF according to Brambilla *et al.*^[44]. There have been a number of efforts to adapt WebML with MDA/MOF/UML family by Brambilla *et al.*^[44], Moreno *et al.*^[45–16] and Schauerhuber *et al.*^[47]. These suggest that it is an advantage to design a model that is UML-based.

5. Only some of the existing models such as those found in Refs. [3, 39–40] provide partial support to Web 2.0.

6. Finally, despite several approaches to incorporating requirements in the web engineering process^[73–80], the existing models still show little integrity with higher-level requirements.

As a concluding note, it is worth mentioning that WebML-based approaches present the most comprehensive way of data modeling for web applications. Further, WebRatio^[9] a WebML-based tool, generates fully executable code. Thus, it is critical to distinguish our work from WebML. A major difference with WebML is the way we treat Web 2.0. While WebRatio enables script units, where Ajax code for Web 2.0 support can be injected, our approach has a built-in support of Web 2.0. This means, using WebML, one needs to manually create the Web 2.0 features required for every component of the web; using our meta-model, however, Web 2.0 features such as search, feedback and update are automatically assigned to all elements. It is also worth remembering that despite the efforts^[44–47], WebML has remained a non-UML language and WebRatio has chosen not to support UML export/import^[99]. Finally, WebML is specifically designed for building web applications. Therefore, it may not be used for specifying applications at a level that is web-independent. A subset of our meta-model is web-independent and is used for PIM level specifications. This enables a minimal input language that is easier to use and is mapped to our meta-model using a set of automated transformations as described in Section 6.1.

4 The Meta-Model

We have created a meta-model that supports all the requirements listed in Section 2.1. This meta-model extends the Botterweck model^[17]. The structure and components of the UI model as well as the relationship between the UI model and data of the Botterweck model have been retained in our model. The parts concerning aspects related to behaviour have been customized. We will revisit the Botterweck

model in Section 4.4 in order to summarize a list of enhancements our meta-model suggests comparing to the Botterweck model.

Figure 1 shows the general structure of the application in our model. An application has several use cases, a number of these use cases may be startup use cases. A number of 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. The meta-model is supplied with a default login use case. This use case includes another default use case, Show Homepage; and is extended by use case, Lock Account. Login use case allows a certain number of login attempts that are modelled through an element of type Page Variable. The behaviour of use cases is modeled through state machines.

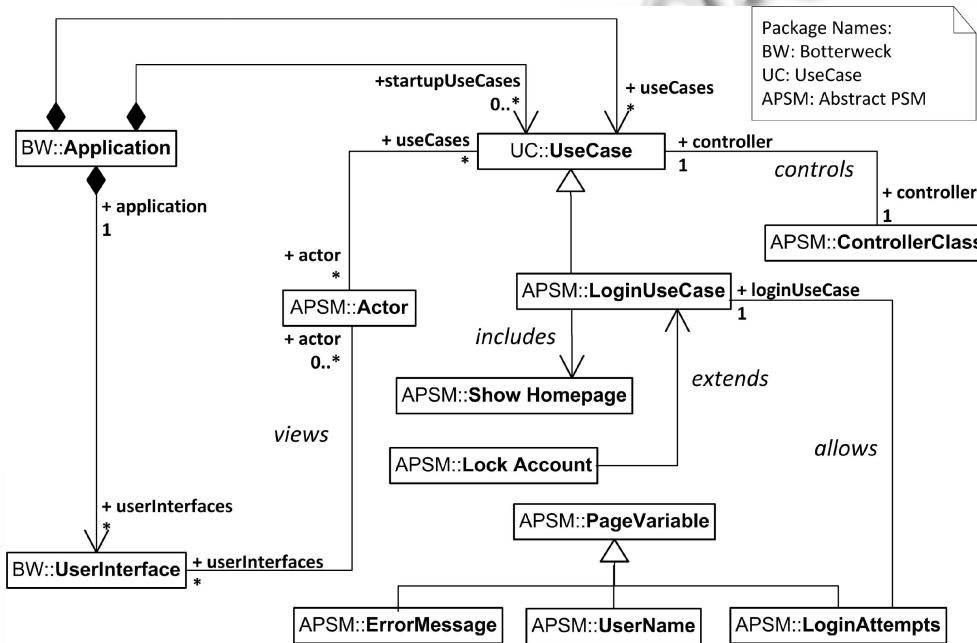


Figure 1. The meta-model of Web information systems, general structure

Figure 2 presents the meta-model of state machines in our approach. Most of the elements in this picture are imported from the UML specification. We added some associations to explicitly connect use cases to state machines. States are associated with use cases; this will be understood as either an inclusion or extension depending on the situation. Also for navigation, such as for instance, a situation where a successful completion of a task leads to another web page, it is necessary to indicate that a given use case sequentially follows another use case. This sequencing of use cases is indicated by attaching use cases to final states. When processing flows from use case *uc1* to use case *uc2*, the final state of the state machine description of *uc1* is associated with *uc2*. This technique for sequencing is also used to forward processing to predefined state machines that describe re-occurring functions.

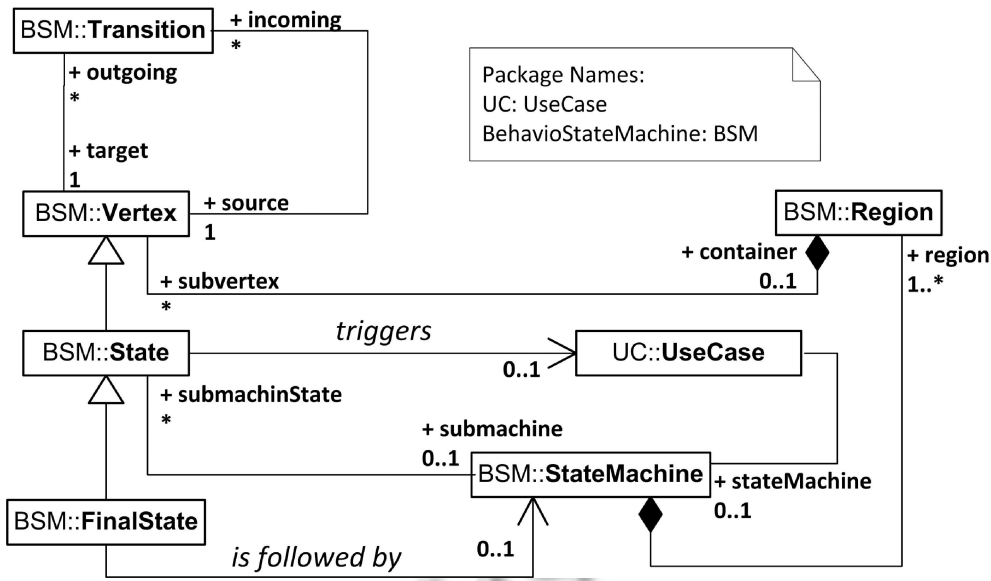


Figure 2. The meta-model of Web information systems, use cases and state machines

Figure 3 shows how our meta-model supports the presentation and data in one model. States may be associated with presentations; we refer to such states as “presentation states”. A presentation is a special UI Composite, which means it could contain other UI Components. UI Components may be associated through a *Field-Operation*. This allows client-side operations to be defined. Examples of such actions are given in the enumeration type *ActionType*. 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 basis for data selection from another data composite. For example, such an association would be created to model a situation where selecting a country in a component affects a list of provinces in another component.

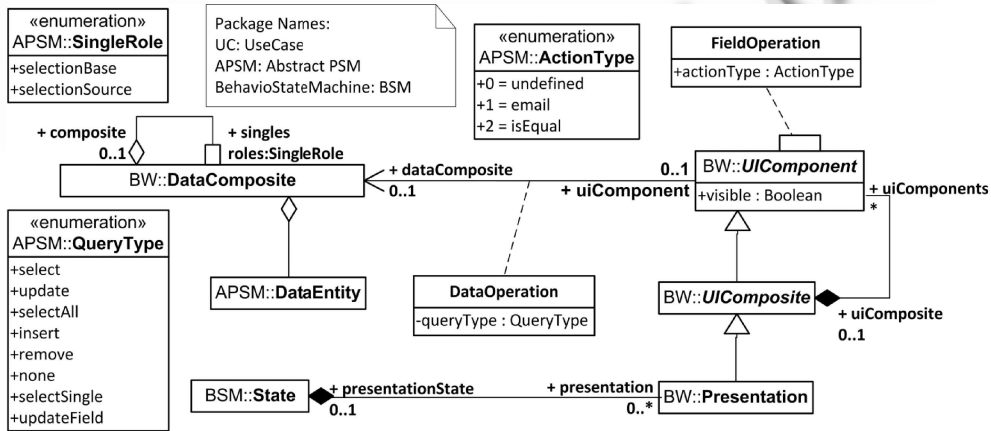


Figure 3. The meta-model of Web information systems, presentation vs. data

Figure 4 presents the part of the model, where events and operations are handled. A controller class is used to control an application according to the behavior defined in a use case based on the use-case controller pattern^[13]. In addition to controller classes, one or more service classes are used to perform operations required for data access. Various specific platforms may use different mechanisms for concrete controller and data access service classes. For instance, one class equivalent to each controller and service would be created for AndromDA, while for WebRatio, controller and service operations are distributed amongst different operation units. At the code level, different strategies are typically used for implementing controllers such as Java servlets or .Net front-controllers. Similarly, services can be compared to DAL files in .Net or Entity Beans in Java.

A service is associated to a data composite. Service operations are basically Create-Read-Update-Delete (CRUD) operations for data composites but the meta-model allows the addition of more complex CRUD operations as well. Such operations must be defined as a new literal added to the enumerative type *QueryType* (Fig. 3). Attribute *crudNature* of the class *Operation* specifies the type of service operation and is used for code generation. An association from class *Operation* to itself models the call structure from controller operations to service operations. A special UI composite, *OperationTrigger*, is used to represent information submitted to the server side, for example, as part of a web form. Operation trigger may cause either an included use case or a controller operation to be triggered.

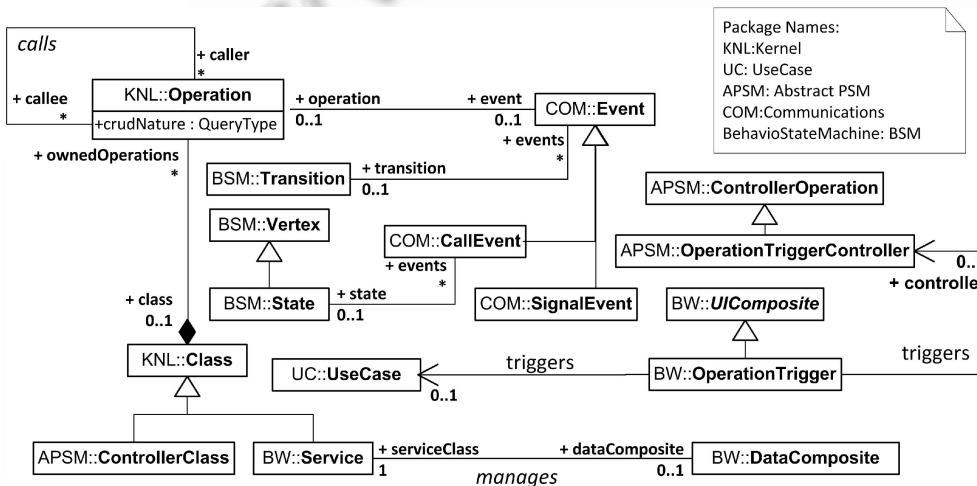


Figure 4. The meta-model of Web information systems, events and operations

Finally, Figure 5 presents the fact that content oriented UI elements, including text, image and video, may have a feedback, which could be another content oriented UI element; that is a feedback may receive feedback as well. Feedback is defined as an interface to allow target specific platforms implementing their own specific strategy. This feature plus the fact that UI composites can recursively contain other UI composites - shown in Fig. 3 - help support Web 2.0 applications. The class *ContentOrientedUIElement* is an abstract class.

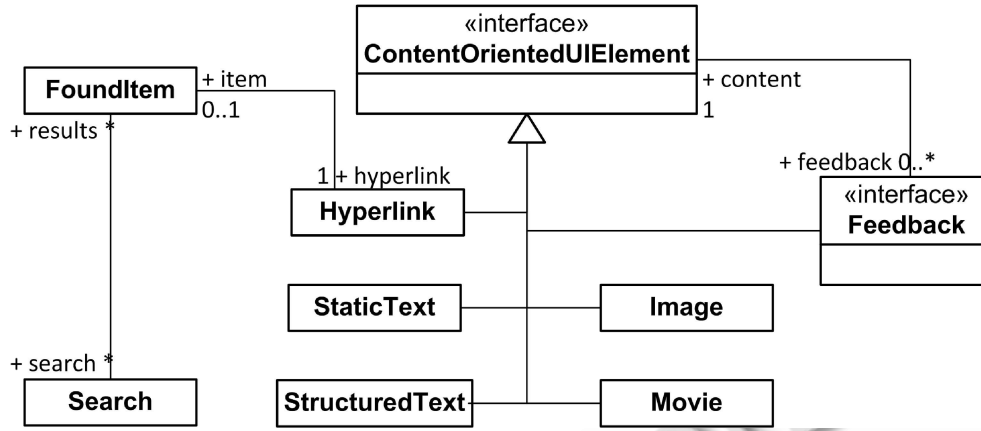


Figure 5. Making the content interactive

4.1 A data-driven case

Figure 6 shows a model for a shopping cart application. The *ShoppingCartController* handles the operation trigger *Make Payment* submitted from a web page. The *makePaymentOperation* performs several tasks among which are an update of the inventory and a record of payment. This is modelled by having *makePaymentOperation* associated to two operations from *ProductService* and *PaymentService* as callee. These operations accomplish the required CRUD operations.

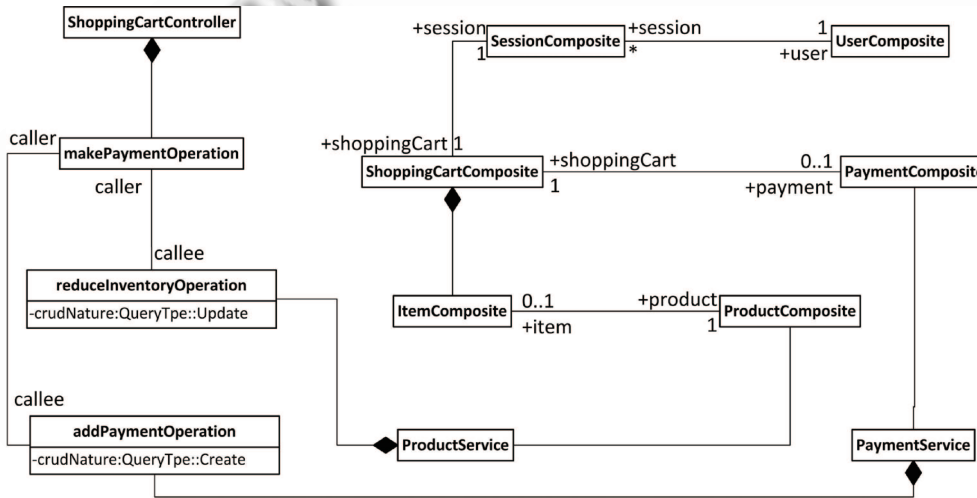


Figure 6. A shopping cart example: make payment operation














4.2 A multi-layered model

The meta-model does not limit the number of layers. For example, the presentation layer could be divided to more than one layer because the UI is a type of UI Composite, which may contain other UI Composites. Both the top-level UI composite and the included ones can be associated with data. Therefore, the top-level presentation layer includes several sub-layers of data, presentation and controllers.

Figure 8 shows the subset of the meta-model used to specify the visual input language. Comparing to the meta-model as the APSM, the model of Fig. 8 serves as the PIM. The figure uses abbreviated package names as follows:

- BW: Elements borrowed from the Botterweck Model
- APSM: Elements added by our meta-model
- BSM: Elements from the UML package, Behaviour State Machines
- UC: Elements from the UML package, Use Cases
- COM: Elements from the UML package, Communications

Table 2 A graphical input language

Symbol	Description
	A Submit Button
	Create Data Query or Add Data to UI Composite
	Data Composite
	Date Display Component
	Hyperlink
	Load All Query or All Data from a UI Composite
	Filtered Version of the above
	Page/Session variable
	Password Input
	Plain Text Input
	A Table Column
	A UI Composite, i.e. Table, Presentation, Trigger
	Update Data Query or All Data to a UI Composite

4.4 Extending the botterweck model

Certain parts of the Botterweck model remain intact in our meta-model. These are mainly the elements that build the contents of the web application such as the input or multimedia elements. Other parts require changes. For example, the connection of data and UI elements is only provided for elements such as *Select* components in Botterweck's model while we require more freedom in this area. Several elements other than *Select* components on a web page may be populated by data supplied from a data source; some examples are text descriptions of items in a shop, pictures, feedback on textual posts and radio buttons. Thus, we choose to assign data to all UI components not only the select ones. This, however, does not eliminate other common features of web applications; as asserted before all features may be developed using the meta-model, the extra specific features are provided as optional add-ons for supplying the automation of the methods. For more detailed specification of the Botterweck model and our extension mechanism see Appendix A.

Compared to the Botterweck model, our meta-model facilitates the data access mechanism by associating UI components to data composites. Also, the data service types are suggested as an association class, which provides an easier way for assigning the proper data service to each UI element. Other changes are performed for supporting specific use cases such as Login and Startup. Another important alteration is to unify the definition of a presentation state with the element *state* as an association attribute. Finally, certain structures are added to support specific features of web information systems such as frequently asked questions and shopping carts. More changes are as follows:

1. Data Associations: A new concept is added to the UI model to support the required data for UI components. The developer is required to know what data are associated with each component. Alternatively the developer may choose to manually add modeling elements required to support a component. This feature is not supported in the original abstract model to the required extent. Some data features can be associated with input fields such as select lists but not to other elements. We require the possibility to associate data elements to other elements such as operation triggers in order to be able to automate the process of generating data-related operations and services. In our meta-model, a UI component could have a data composite associated with it. This association is attributed with the query type, which could be one of the five operation types.

2. Operation Triggers: Triggers need to be associated with data composites in order to enable the automated development of required data services and processes. However, the original model considers operation triggers as UI elements as shown in Fig. 5, which makes it impossible to project this aspect. We change this by having operation triggers inherited from *UIComposite*. This is reasonable because in practice a web operation trigger, when submitted, is wrapped with the parameters equivalent to the UI elements within the submit form. In the original model, a Trigger was a *UIElement*, which is changed to a *UIComposite* in the current model. Note that since a *UIComposite* is a special kind of a *UIComponent*, the meta-model still allows designing a Trigger as a plain component with no other components wrapped in its model. Another change was required to denote the order of events on a presentation unit. Presentation units could hold several kinds of events that are not necessarily

operation triggers (i.e. submit button). For example, drop-down select components can cause a selection event that affects the contents of other components. In order to enable the description of such behaviors, we have added a new association between *DataComposite* and *UIComponent* elements. The *eventHolder* can be attributed with an order property. This property is only used to specify the order of such events when more than one component can cause a page to call a controller operation.

3. Cross-Referencing State Machines: UML allows states to enclose other state machines. This capability could be used to fulfill a variety of motivations. A state enclosing another state machine can be used as an indication of an extend/include relationship; that is the use case represented by the enclosed state machine is either an extension to the use case of the original state machine or is included within the original use case. The relationship between *State* and *StateMachine* is borrowed from UML but the relationship from *FinalState* to *StateMachine* is specific to our meta-model to support the sequence of flows. An operation trigger may also accept a reference to another use case as a denotation of an included use case. This is called the *includedUseCase*. The sequencing of state machines is indicated by the attachment of the final state to the use case corresponding to the state machine activated afterward; this is performed for navigation reasons – such as situations that successful submission of a task results in navigating to another web page – or to supply the possibility of forwarding the process to an automatically added state machine such as *Login/Logout* state machines.

4. Page/Session Variables: Page variables are session/page parameters that carry information either to be shown or processed in a web page, or regarding controller operations. We add page variables to operations aimed at a presentation state in the circumstances such as the followings:

- To carry a default error message when the outgoing transition from a choice is labeled false.
- For every login-required presentation state, we make it mandatory for all incoming and outgoing transitions/events as well as regular states before and ahead to accept a page variable carrying a username. One must note that this is by no means an authentication technique that must be used as such in the implementation. It is rather an abstract indication of the use of a session variable for authentication. Different concrete implementation procedures are used by different specific platforms.
- A page variable named *loginAttempts* is created when a login-required state is bound to a maximum number of login attempts.

5. Hidden Components: It is considered that UI components may be hidden. To this end, the class *UIComponent* is equipped with an attribute “visible” of type Boolean. There is no functional difference between page variables and hidden components, but more a design decision the developers will take. It is important to note that page variables are only used to pass parameters from one presentation unit to another one; they may not be shown to the User but UI components are created on the target page – of a transition - and may or may not be shown to the User.

6. Security Features: Certain aspects of web applications require secure attributes that prevent insecure access to the data, presentation and operations. The

implementation of this feature may vary in each specific platform. Our meta-model defines an interface named *SecureItem* in order to support items that may require secure access. By default, the three elements, *UIComponent*, *DataComposite* and *ControllerClass* implement this feature, which may or may not lead to explicit usage of security features by the developer according to the requirements and platform-based decisions.

7. Web-Specific Features: Specific features such as *Frequently Asked Questions (FAQ)*, *Search*, *Shopping Carts*, *Login/Logout*, *Site Maps* and *Feedbacks* are added to the Botterweck model.

8. Other Changes that are noteworthy are as follows:

- a. A concrete association between a data composite and the corresponding service class that includes the type of data operation is required to support data storage, retrieval and processing tasks of web information systems.
- b. Every application requires one main use case to launch the process. We refer to this as the startup use case.
- c. The behaviour of use cases is handled by use case controllers. The model needs a simple change to handle use case controllers.

5 Examples

The first example is chosen from a case study that is a Web 2.0 application, *Dilmaj*^[49]. *Dilmaj* is a web-based, collaborative, multilingual project about language and language development supported by a group of developers. Many domestic languages are challenged in modern times. The goal of *Dilmaj* is to offer a central place where users can contribute to the development of a chosen language, and its relation to other languages. Users may join, suggest terms, and comment on others' suggestions or rank entries. Figure 9 shows a use case diagram that includes several use cases of *Dilmaj*. The diagram contains main use cases that represent pivotal functionalities of the application. The three main actors are User, Member, i.e. a registered User and Admin.

Figure 10 shows the input model of use case *Add Comment* defined using the language of Section 4.3. A complete compilation of this application using the APSM meta-model is presented in Ref. [50].

Figure 11 shows an object model instantiated from our meta-model and corresponding to the input model in Fig. 10. Transitions are not included for the sake of clarity. The model involves three of the application use cases: *Add Comment*, *View Comments* and *View Dilmaj*. Use case *View Comments* is attached to the state *Add Comment* of state machine to realize the inclusion relation shown in Fig. 10. Use case *View Dilmaj* sequentially follows use case *View Comments* as expressed by the attachment of use case *View Dilmaj* to the final state of use case *Add Comment* state model. Note that there is a concrete link between every pair of name-dependent objects. This addresses issue 3 in Section 2.1 about name-based dependencies. The object model also shows that the input from *Add Term*, the table *Terms Table* and the button *Close* are all considered independent parts of the presentation to ensure compatibility with web 2.0.

6 Using the Meta-Model along with Model-Driven Transformations

In order to evaluate the meta-model for specifying an abstract web application in a model-driven context, we have developed four sets of mappings:

1. PIM-to-APSM: This set maps the input model such as the one in Figs. 12-13 to the web model such as the ones of Figs. 15-16. Since an objective of model-driven development is to automate parts of the software engineering process, it is important to show it is possible to derive the full specification of an application from a minimum set of descriptions.

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.

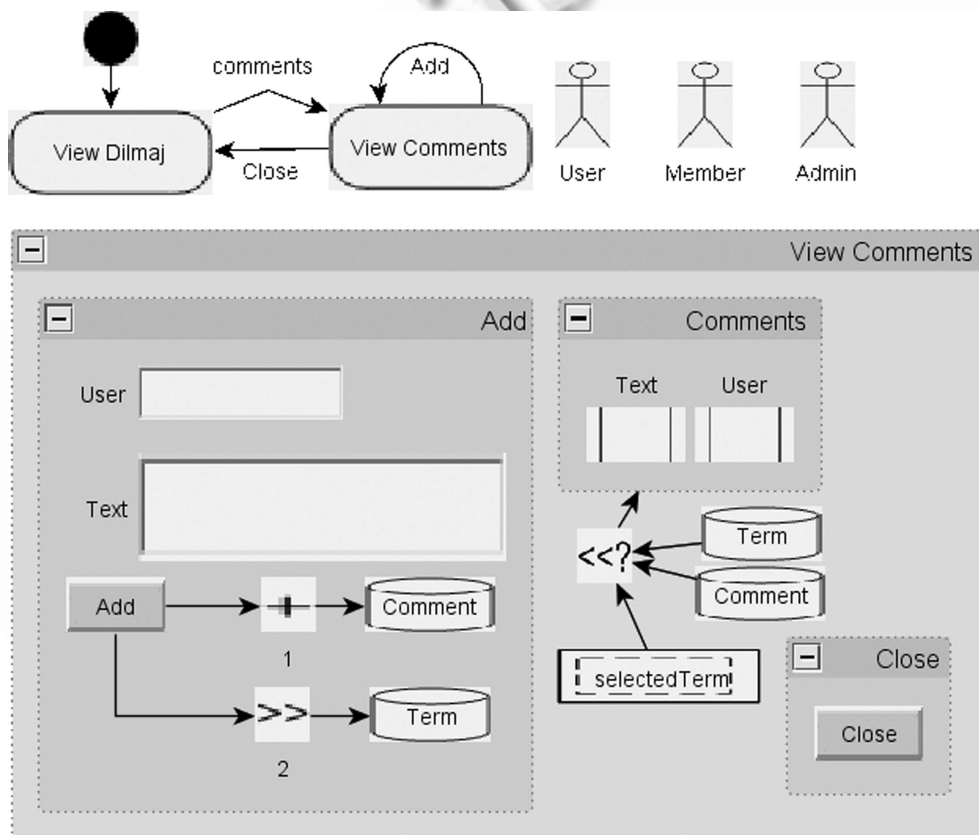


Figure 10. The PIM of the use case, add comment

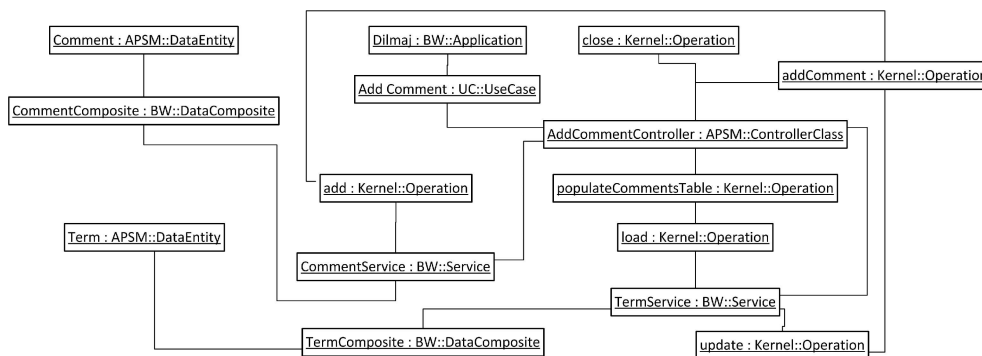


Figure 11. The object model of Fig. 10

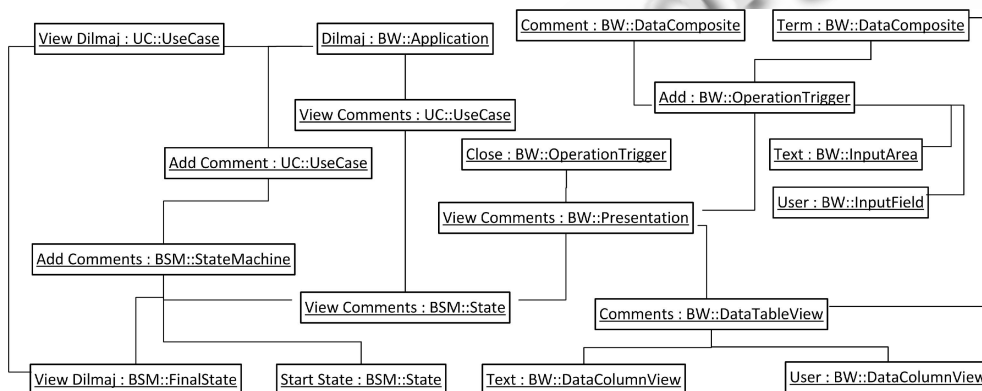


Figure 12. The object model of behavioural features of Fig. 10

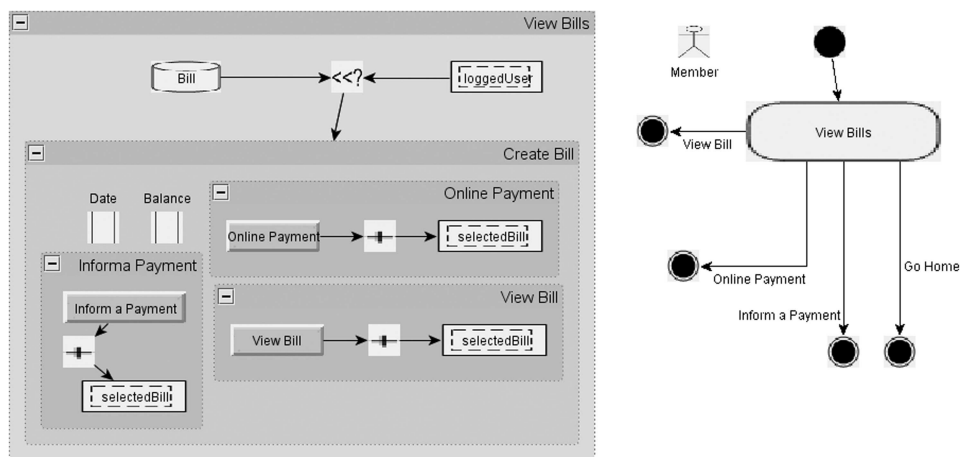


Figure 13. The PIM of the use case, view bills of the AMS

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

These mappings are expressed using QVT relations^[51] and are listed very briefly in this section. The complete listing of the QVT relations for the above four sets of mappings is provided in Ref. [52].

6.1 PIM-to-APSM

This set creates a full web application from an input model in the visual language defined in Section 4.3. These mappings ensure that the elements of the meta-model, which are used to define the usability and data requirements, are properly integrated with the ones specifying the behaviour and data access mechanisms. Figure 17 presents a summary of the information used from the PIM as well as those created at the APSM level.

- Data objects and navigation paths through data objects are created based on data associations found within the UI model.

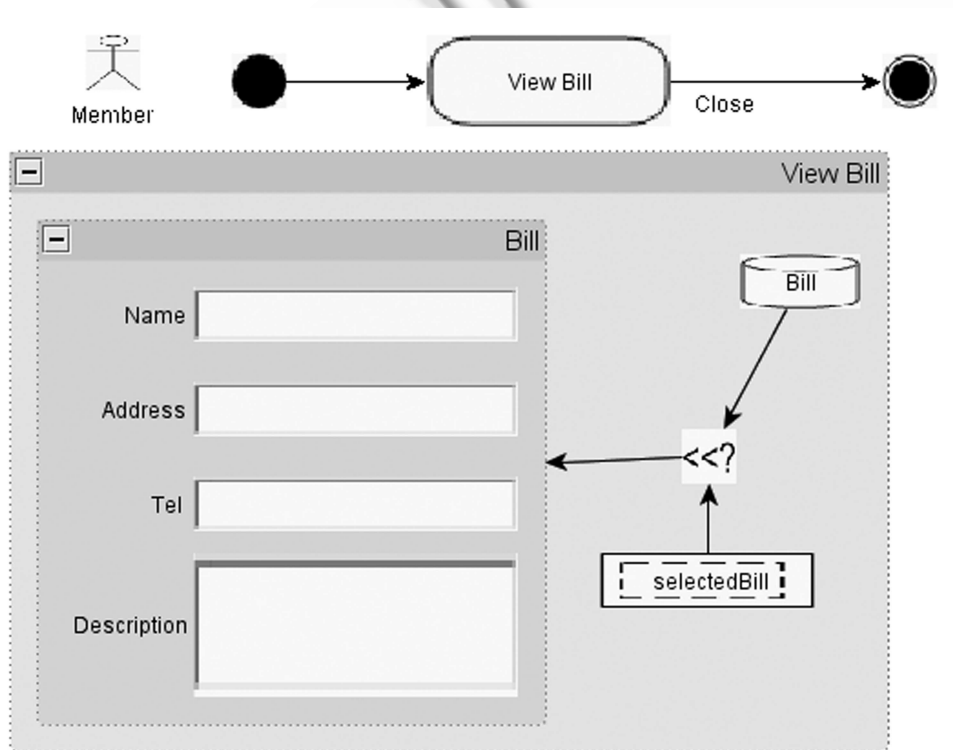


Figure 14. The PIM of the use case, view a bill of the AMS

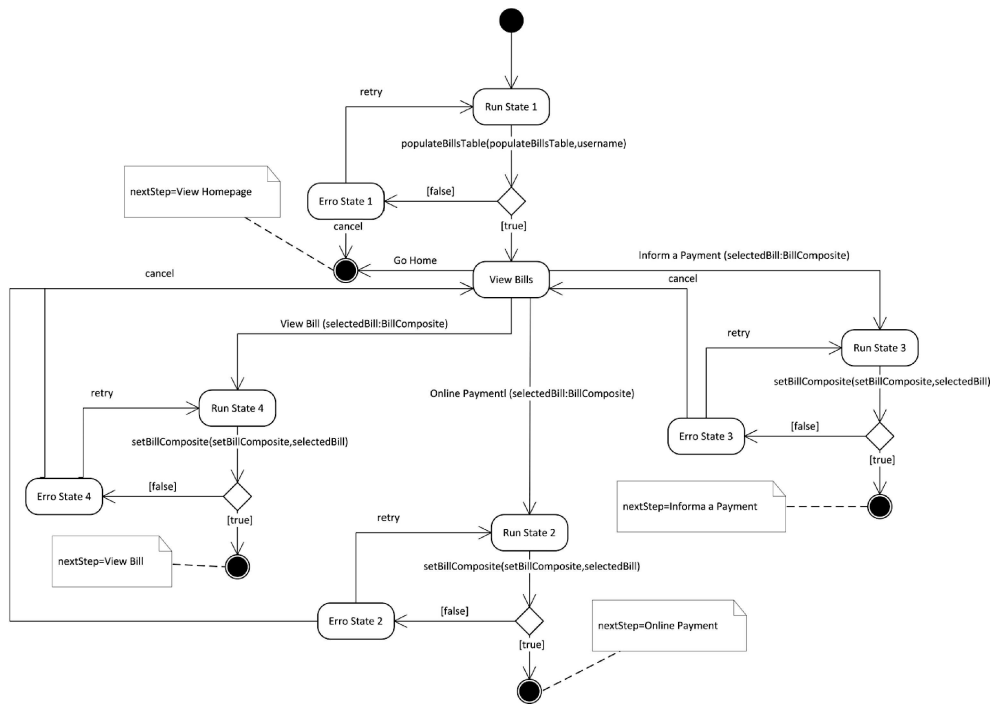


Figure 15. The APSM of the use case, view bills of the AMS

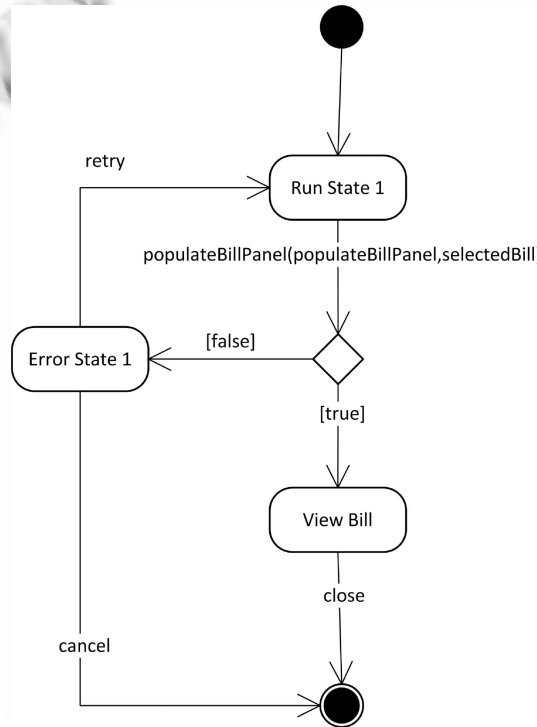


Figure 16. The APSM of the use case, view bill of the AMS

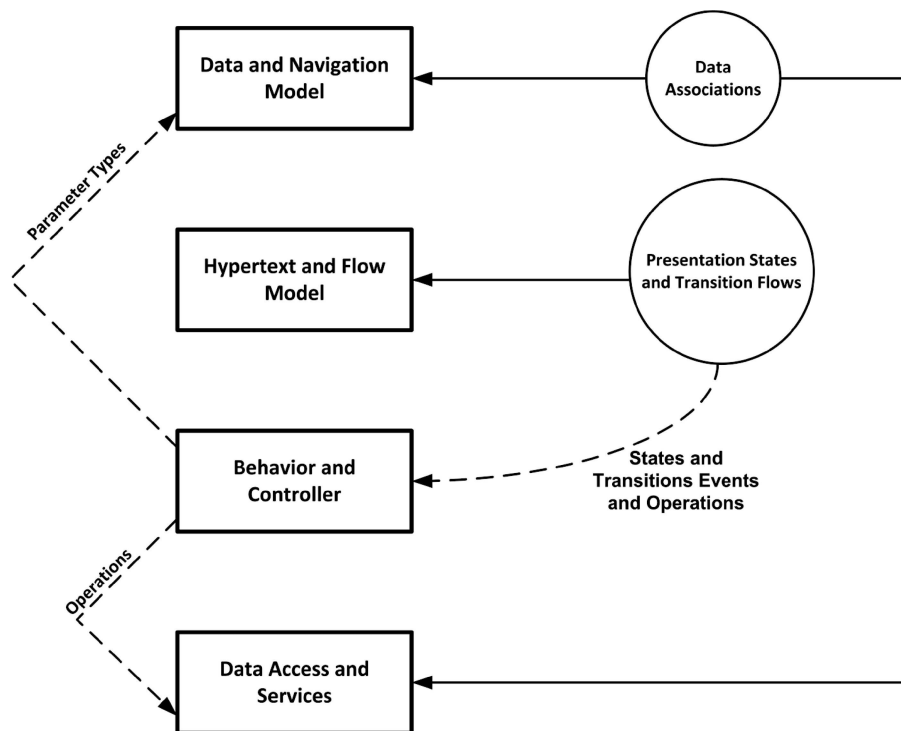


Figure 17. Summary of PIM-to-APSM mappings

- The contents and 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 behavior 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.

An example of such mappings was given in Section 4. The object model of Fig. 11 is a result of applying the PIM-to-APSM mappings to the object model of Fig. 10.

6.2 APSM-to-AndroMDA

AndroMDA is a model-driven development framework that allows automatic code generation for web applications. AndroMDA accepts UML models enclosing transformation tags and modelling stereotypes. An AndroMDA application is composed of several use cases. The behaviour of use cases is modelled using state machines. States of state machine may represent presentation or behavioural features of use cases. Every use case is controlled by a controller, where events of states and transitions defer operations to. Controllers forward calls to service classes that are associated with data elements. A special type of data objects, *Value Object* is used to transfer information between services, controllers and the front-end. We briefly describe a mapping to an AndroMDA-specific platform. Following are some definitions regarding this platform:

- A deferrable event is an event that invokes a controller operation. Deferrable events are used to assign call events to operations with states.

- A value object is an object that carries information between domain objects and the presentation or data access layer.
- *FrontEndView* is a state stereotype used for states that represent web pages.
- A signal event is an event that is usually carried by an incoming transition to a front-end state. A signal event carries output fields to be shown.

The following rules apply to all models:

- 1) There must be one controller class per use case.
- 2) There must be a service class per data object.
- 3) Controller classes must be dependent on their objects' service classes.

These additional rules are specific mapping rules for an APSM model to an AndroMDA-specific PSM:

- 4) Every presentation state is mapped to a state stereotyped as *FrontEndView*.
- 5) For each operation trigger:
 - a) A signal event is created on the outgoing transition.
 - b) A deferrable event is created on the next state. If the next state is a choice then the deferrable event is created on the transition ending to the choice.
 - c) A controller operation is generated to be called by the generated deferrable event.
 - d) The set of input parameters for the signal/deferrable events as well as the controller operation are created based on the UI components belonging to the operation trigger.
 - e) For every domain object referred to by a component of an operation trigger an entity domain object and a value object are created. If the domain object requires one of operations update, delete, or insert then the tag *Manageable* is added to the target object, to force AndroMDA to generate the corresponding operations. Also:
 - i) A dependency from every entity domain object to the relevant value object is created.
 - ii) An operation is added to the service class to perform the corresponding CRUD operation.
 - iii) A call is added to the controller operation to call the service.
 - iv) A dependency is added from the service class to the domain object so that the service class can access the instances of the domain object.

6) Every UI component not owned by a trigger becomes a parameter in the set of parameters on a signal event belonging to the incoming transition.

7) Every choice in the state machine results in the creation of a controller operation that returns a value, based on which the transition to be taken is decided.

More details are reported by Fatolahi et al.^[53–54].

6.3 APSM-to-WebRatio

We examined the meta-model through non-refining mappings as well; that is mappings to non-UML-based models. A good example of a non UML-based, and yet

very popular model is WebML. We have created a set of mappings from APSM to a specific configuration of WebRatio which is a WebML-based tool. WebML supports a hierarchy of Site View, Area, and Sub-Areas or Pages recursively. Modeling units are either content units that are devoted to contain hypertext content or operation units that facilitate the controlling operation and data-related operations. Pages could be categorized as:

- Homepage: the default page of a website
- Default Page: the default page of each area
- Landmark Page: pages that are accessible from every other page such as quick links

links

The following is a summary of the APSM-to-WebRatio mapping:

1) Site View: Actors are mapped to site views.

a) For every site view, a default homepage is created named after the view name following by the term 'Home'

b) For every use case related to the actor

i) For every state machine of the use case

(1) A page is created named after the first presentation state of the state machine

(2) A link from the default homepage to the page created in previous step is occupied

2) Entry Units: For every signal event found on a transition exiting a presentation state an entry unit is inserted within the page equivalent to the presentation state

a) For every data operation associated with the signal event, the entry unit would be paired with an operation unit. The type of the operation unit depends on the type of the data operation. For example, a data create operation would be mapped to a create unit.

3) Operation Units: For every pair of a signal-event/call-event found in the state machine, the map of the signal event, i.e. the operation unit needs to be supplied by the followings:

a) Fields that are created based on the call event parameters

b) An OK link that leads the flow toward a target page in case the operation is successful. The target page could be the mapping of 1) another presentation state, 2) the homepage of the view or 3) the first presentation state of another state machine based on the following:

i) The first case is taken when the transition guarded with a true condition exiting the choice after the call event leads to a presentation state.

ii) The second case is selected if the transition ends at the final state.

iii) The third case is chosen when the transition ends at the final state and the final state is associated with another use case

- c) A KO link that leads the flow towards an error page or message in case the operation fails.

6.4 *APSM-to-GWT*

Google Web Toolkit (GWT) is a development framework mainly for client-side design of Web 2.0 applications using an asynchronous service-based architecture. GWT extends Java and supports easy design and installation of UI components. The GWT itself does not support the Java Database Connectivity (JDBC)^[56] but it is possible to use JDBC to connect an RDBMS. We have, however, chosen to use the Java Data Objects (JDO)^[57] mechanism to support data-related features of the platform. As a result, the application is fully deployable on the Google web server. A typical GWT application has a server and client package as well as a shared package. The client package includes the definition of the page UI components while the server side implements the services. Our configuration allows one service for each data composite. Although it is possible to develop Web 2.0 applications using our two other specific platforms, the mapping to GWT provides a more convincing evaluation regarding Web 2.0. The following is a summary of the APSM-to-GWT mapping:

1. Data entities are mapped to Java Persistent Objects. These are only used for storing data or retrieving them from the server
2. Data Composites are mapped to serialized classes within the shared package, data composites are used to transfer data as parameters of the service operations.
3. For every data composite a service will be created, the service interface and its asynchronous map are defined at the client side while the implementation is defined at the server.
4. Only one page is defined but every presentation state will be mapped to a panel. This is also true about different actors' views
5. Every signal event is mapped to a click handler and a key handler. These will forward signal events to functions defined in a controller class. This controller class decides which services to refer to for the accomplishment of the tasks. This will form the mapping of call events.
6. A one-to-one mapping between UI components of APSM and the widgets of GWT is defined.
7. A one-to-one mapping is also defined between data types. Arrays are mapped to Java List type and these Lists are instantiated as - ArrayLists whenever required.

Figure 18 shows a high-level visualization of the mapping in this section. As this figure shows, a panel is assigned to every actor. This supports the definition of different views for different actors. All other pages and forms are defined as sub-panels of these panels.

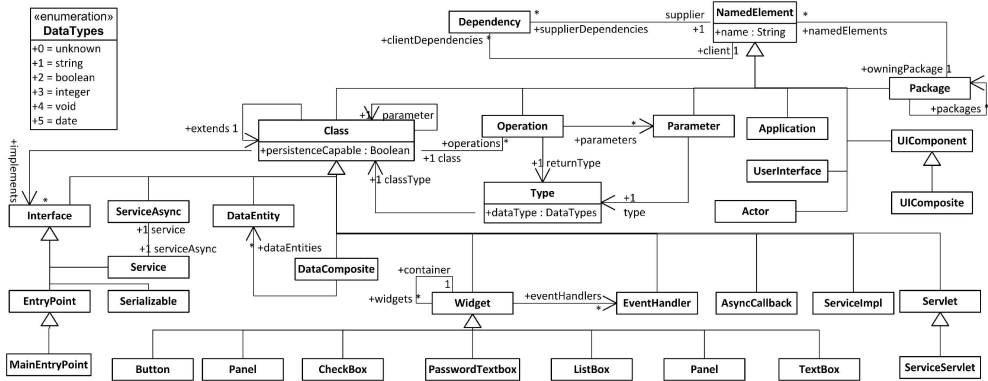


Figure 18. APSM-GWT mapping: a high-level viewpoint

7 Evaluation

Our meta-model successfully satisfies the requirements discussed in Section 2:

- Unlike the existing models, we provide operational support from presentation layer to data-access layer facilitated by controllers that dispatch triggers throughout different data services. The transmission of data between layers is supported by data composites while the data itself is modeled using data entities. As shown in Section 4.2, our model supports a multi-layered MVC architecture.

- Our model remains at the APSM level and is yet adaptable to other SPSMs; this ensures platform-independence. Our mappings to specific platforms show that the meta-model can map to multiple platforms. The tested platforms belong to three different categories. We were able to support specific platforms with different features than the common selected ones using our transformations. For example, WebRatio and GWT do not specifically support the controller pattern common; yet our transformations were able to reformat the controller scheme of the APSM to WebRatio (as distributed operation units), as well as to GWT (as controller operations accessed through entry points).

- We use concrete associations between different elements to ensure the required traceability by MDD transformations independent of naming. For example, the concrete association instance between a *UIComponent* and a *DataComposite* in Fig. 3 provides the same type of support as suggested by Cichetti and Di Ruscio^[82] in terms of an association from *DataCompositionWLink* and *CompositionWElement*.

- Our model is an extension to another UML-based model^[17], which has been designed following the UML extension mechanism.

- Our model pays specific attention to several features of web information systems such as authentication and session mechanisms, which are not widely handled in other approaches.

- Another contribution is the consideration of elements that help developing applications that embody Web 2.0 features. Two main features of our model that help support Web 2.0 are 1) a presentation state may be composed of several presentation units, which allows independent update of different units within the same page; and 2) content-oriented UI elements accept interactions such as comments, feedback

and ranks, as required by Web 2.0. Furthermore, content-oriented UI elements are attached with search interfaces that imply a fully searchable content.

- Our approach closely relates requirements to lower-level models and hence increases the chance of developing applications in accordance with requirements.

- Use cases, and their behaviour, are modeled using state machines. Several features are provided to suggest mappings from different use case structures such as inclusion/extension, use case steps and scenarios directly to state machines. The resultant APSM state machines inherit the same type of relationships that exist among use cases.
- We consider UI prototype as a requirement that should be defined as an input while some other approaches try to generate the UI as an output, e.g. Wu et al.^[7]. Our position is that the desired UI model should be seen as a requirement – particularly for web-based information systems.
- We also include information elements within the description of UI prototypes as well as the query marks used to elaborate the relationships between a UI element and associated information elements that are used for the automated generation of data-related operations.

The transformations were implemented using MediniQVT^[60]. In order to do this, we needed models in the form of ECORE^[61]. Thus, the Eclipse Modeling Framework^[61] was used to generate the meta-models. In order to define the input, an open-source designer, *yEd*^[62] was used. A utility to transform the *yEd* files to EMF was created. Once the transformations are executed in MediniQVT, the output EMFs may be transformed to specific platforms using other utilities that perform file conversion transformations. The implemented approach is named MODEWIS, which stands for Model-Driven Development of Web Information Systems. Figure 19 shows how MODEWIS interacts with other components in this implementation.

A utility, *MODEWIS_PIM_Generator* transforms the *yEd* files to EMF^[63]. Once the transformations are executed in MediniQVT, the output EMFs may be transformed to specific platforms using other utilities that perform file conversion transformations such as *MODEWIS_GWT_Generator*^[64]. A VBA module, *MODEWIS_ProofRead*^[65] has been implemented under Microsoft Visio^[66] to visualize the results of transformations and hence to verify their correctness and validity. A typical scenario for working with MODEWIS – e.g. targeting GWT - is as follows:

1. Create the PIM according to the meta-model of Fig. 8 using *yEd*^[62]. A useful set of symbols for defining the PIM using a visual language is available online^[95].
2. Transform the *yEd* file to EMF using *MODEWIS_PIM_Generator*^[63].
3. Map the resultant PIM to APSM using *PIM_APSM* QVT relations^[52].
4. (Optional) Check the validity of the APSM using the APSM tab of *MODEWIS_ProofRead*^[65].
5. Map the resultant APSM to GWT using *APSM_GWT* QVT relations^[52].
6. (Optional) Check the validity of the GWT model using the GWT tab of *MODEWIS_ProofRead*^[65].
7. Transform the GWT model to GWT code using *MODEWIS_GWT_Generator*^[64].

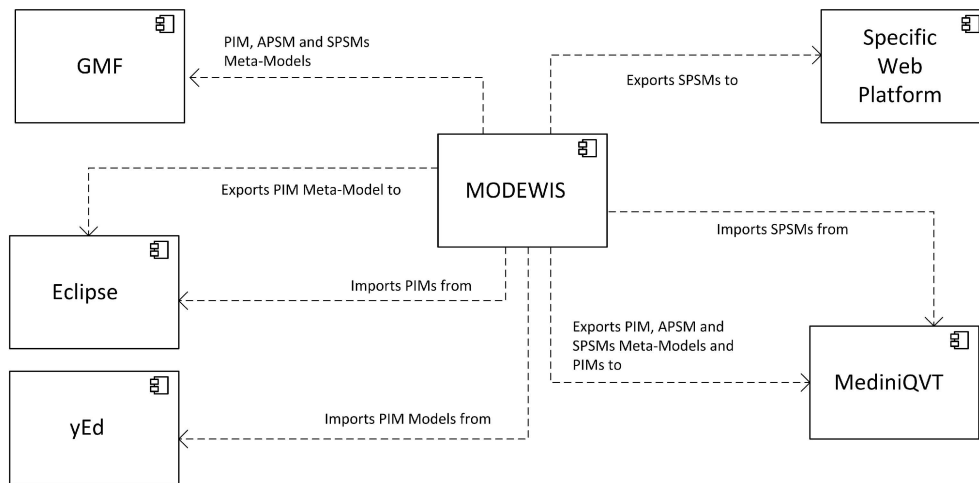


Figure 19. The architecture of MODEWIS

The meta-model have been applied to various case studies beside Dilmaj. Examples include the Election Management System (EMS)^[58], a Team Management System (TMS)^[59], and an Account Management System (AMS)^[94] that is a project foreseen by a waste/water management company to replace an existing non-web system. Case studies have been selected based on several parameters such as the complexity of CRUD operations, number of use cases, complexity of the use case model in terms of number of inclusion/extensions and the nature of the domain. For example, a case study that implements a web-based version of a cash/points card for restaurant retailers was recently added. This case study covers fewer use cases compared to our other case studies but implements several situations where an operation involves more than one instance of one entity in the same page.

We have compared an automatically generated TMS application with students' projects delivered as part of their course projects. The comparison shows that in most cases the generated application is equal to what students have manually developed in terms of functionality. Another evaluation was conducted with the AMS. This evaluation is different because there is an existing system in use. The developers were gathered in a workshop. They were provided with a brief introduction of the approach as well as samples outputs. Then they were asked to evaluate how much extra work would be required to complete a system automatically generated using MODEWIS. The result was up to 20% in terms of extra lines-of-code would have to be added after generation. However, most of the extra-code concerns the page design. Finally, the evaluation of MODEWIS in case of Dilmaj was formed as a group development work. The experience resulted in a complete application modeled and coded using MODEWIS except for style sheets. The Dilmaj code is made available online^[96].

In summary, results show that applications developed using our meta-model support the functional requirements, the UI and navigation requirements as well as classes required to model services, data and the application logic^[48]. In some cases, extra coding work might be required to improve the efficiency, enhance the UI and apply more complicated logical behaviours. Other manual interventions might be required to justify the load balancing of data services. Results also show that input models for

Web 2.0 applications are generally composed of more sophisticated UI models while the state machines describing use cases are often simple and small. On the contrary, input models for conventional web applications contain relatively more complicated state machines and simpler UI models for presentations. The development of all case studies, using MODEWIS, was completed in up to one month depending on the complexity of the application as well as the difficulty of designing mappings from the APSM to the target specific platforms.

Different types of transformations are required for every target. The mappings to AndroMDA belong to the category of *Refining* mappings according to Mellor et al.^[55]. Refining mappings are defined between two sets of models both defined based on the same meta-model. Since both APSM and AndroMDA are based on UML, the APSM-to-AndroMDA mappings are refining mappings. The mapping refines the source model by adding more details to it or altering its existing features. Mappings to WebRatio and GWT are more complex because the target has a different structure than the source. These types of transformations may be called *Migrating* transformations – according to Mellor et al.^[55] – that copy the source model to a completely different format at the target.

The visual input language is suggested as a facilitator to enable the usage of the meta-model using a minimal subset and to present a user-friendly way of defining applications (the PIM). Figures 10, 13 and 14 may seem to present a rather complicated language. However, one should notice that, these figures are not directly created as they appear. The actual implementation is made possible using several shortcuts to facilitate the design of the UI and its presentation. For example, as Fig. 20 shows, for defining data operations attached to an element, the user does not need to apply the language as presented in Fig. 10. Instead, drop-down combo boxes are offered to present existing options and to provide a more user-friendly way of defining the operations.

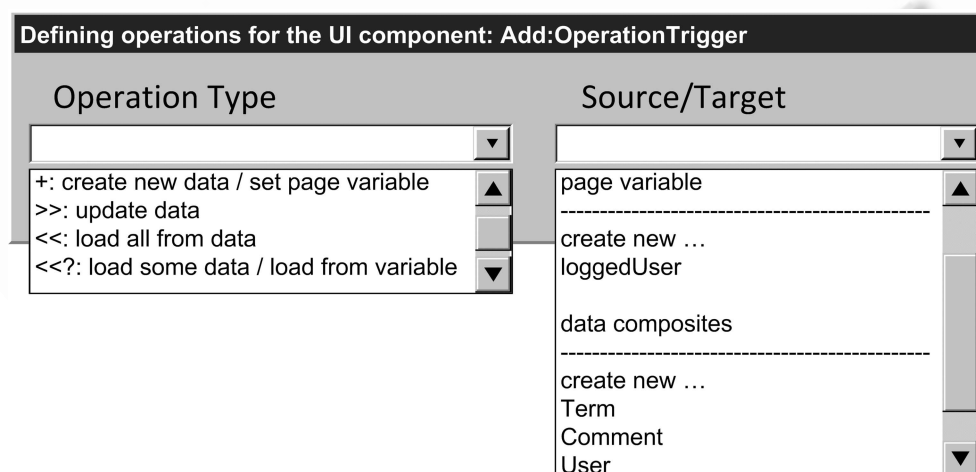


Figure 20. Defining attached operations in practice

8 Conclusion

In this paper, we presented a meta-model for web information systems. This

meta-model is built upon a survey of the existing models used to discover what features a meta-model should bear to effectively support model-driven development of web applications. A selected model was then extended because of the need to cover more concrete features of web information systems. Our meta-model provides elements to model different aspects of a web application at an abstract level. Models instantiated from this meta-model will carry many of the common elements of web applications without being dependent to any specific web platform.

Our meta-model supports abstract models of data-related features both at the UI and service level. It also covers basic features required to specify the functionality expected from Web 2.0 applications. The meta-model helps avoid commonly-found inconsistencies in web models. These inconsistencies can cause ambiguities when used in a model-driven context. Both the input models and transformation to APSMs can be reused to reduce development time. Finally, the integration of functional, usability and data requirements in the meta-model increases the chances of developing applications in accordance with requirements.

Our future work is mainly devoted to developing an overarching application that provides an integrated environment of all implemented utilities supported by a single user-friendly interface. This suite will provide add-on utilities for the developer to customize the meta-model and help develop and deploy platform-specific transformations.

References

- [1] Ceri S, Fraternali P, Bongio A, Brambilla M, Comai S, Matera M. Designing Data-Intensive Web Applications. Morgan Kaufmann. 2006.
- [2] Fatolahi A, Somé SS, Lethbridge TC. A model-driven approach for the semi-automated generation of Web-based applications from requirements. Proc. of International Conference on Software Engineering and Knowledge Engineering. 2008. 619–624.
- [3] Hernández C, Quintero R, Sánchez LZ. Towards the Definition of a Metamodel for the Conceptual Specification of Web Applications Based on Social Networks. ICCSA (2) 2010: 361–369.
- [4] Kavaldjian S. A model-driven approach to generating user interfaces. Proc. of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, Doctoral Symposium. 2007. 603–606.
- [5] Nikolaidou M, Anagnostopoulos D. A systematic approach for configuring Web-based information systems. Journal of Distributed and Parallel Databases, May 2005, 17(3): 267–290.
- [6] Schauerhuber A, Wimmer M, Kapsammer E, Schwinger W, Retschitzegger W. Bridging WebML to model-driven engineering: from document type definitions to meta object facility. IET SOFTWARE, 1-3. 2007. 81–97.
- [7] Wu JH, Shin SS, Chien JL, Chao WS, Hsieh MC. An extended MDA method for user interface modeling and transformation. The 15th European Conference on Information Systems. 2007. 1632–1641.
- [8] AndroMDA, www.andromda.org, 15-02-2007.
- [9] WebRatio, www.webratio.com, 2-2-2008.
- [10] Jr. Whitehead EJ, Ge G, Pan K. Automatic generation of hypertext system repositories: a model driven approach. Hypertext 2004: 205–214.
- [11] Google Web Toolit, Google Code, <http://code.google.com/webtoolkit/>, June 2010.
- [12] OMG. MDA Guide Version 1.0.1, 12th June 2003.
- [13] OMG. Unified Modeling Language: Superstructure, February 2007.
- [14] Ceri S, Fraternali P, Bongio A. Web modeling language (WebML): a modeling language for designing Web sites. Computer Networks, 2000, 33(1-6): 137–157.
- [15] Kroiß C, Koch N. UWE Metamodel and Profile User Guide and Reference, Technical Report,

- February 2008. Available from (www.pst.ifi.lmu.de/projekte/uwe).
- [16] IBM – Rational Unified Process (RUP), <http://www-01.ibm.com/software/awdtools/rup/>, September 2009.
 - [17] Botterweck G. Multi-Front-End-Engineering - Ein modellgetriebener Ansatz zur Entwicklung von Anwendungen mit mehreren Front-Ends [Ph.D. Thesis], Koblenz, Germany: Verlag Dietmar Foelbach, ISBN 978-3934795716. 2007. <http://www.amazon.de/Multi-Front-End-Engineering-modellgetriebener-Entwicklung-Anwendungen-Fronts-Ends/>.
 - [18] Muller PA, Studer P, Bézivin J. Platform Independent Web Application Modeling. “UML” - The Unified Modeling Language. Springer Berlin / Heidelberg. 2003, 2863/2003: 220–233.
 - [19] Baresi L, Colazzo S, Mainetti L, Morasca S. W2000: A Modeling Notation for Complex Web Applications. In: Mendes E, Mosley N, eds. Web Engineering: Theory and Practice of Metrics and Measurement for Web Development. Springer, ISBN: 3-540-28196-7, 2006.
 - [20] Schmidt D, Stal M, Rohnert H, Buschmann F. Pattern-Oriented Software Architecture. Chichester [England]. New York: Wiley, 2000-c2007.
 - [21] He C, Tu W, He K. Role based platform independent web application modeling. Proc. of the Sixth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'05). 411–415.
 - [22] Blankenhorn K. A UML Profile for GUI Layout, Master's Thesis, University of Applied Sciences Furtwangen, Department of Digital Media May 23, 2004.
 - [23] Da Silva PP, Paton NW. Improving UML Support for User Interface Design: A Metric Assessment of UMLi. ICSE Workshop on SE-HCI. 2003. 76–83.
 - [24] Vanderdonckt J. A MDA-compliant environment for developing user interfaces of information systems. Advanced Information Systems Engineering. Springer Berlin/Heidelberg. 2005(3520): 16–31.
 - [25] Schattkowsky T, Lohmann M. Towards employing UML Model Mappings for Platform Independent User Interface Design. Springer Berlin / Heidelberg. Volume 3844/2006, Satellite Events at the MoDELS Conference. 2005. 201–209.
 - [26] Diamodl. <http://www.idi.ntnu.no/~hal/research/diamodl>, July 19, 2008.
 - [27] Li J, Chen J, Chen P. Modeling web application architecture with UML. Proc. of the 36th International Conference on Technology of Object-Oriented Languages and Systems, 2000. TOOLS - Asia 2000. 265–274.
 - [28] Bogdan C, Falb J, Kaindl H, Kavaldjian S, Popp R, Horacek H, Arnautovic E, Szep A. Generating an abstract user interface from a discourse model inspired by human communication. HICSS. 2008. 36.
 - [29] Costa D, Nóbrega L, Nunes NJ. An MDA approach for generating web interfaces with UML ConcurTaskTrees and canonical abstract prototypes. Task Models and Diagrams for Users Interface Design. 137–152.
 - [30] De Souza RAC, de Barros RSM. A Model-Driven Method for the Development of Web Applications User Interaction Layer. TASE 2008. 91–98.
 - [31] Sukaviriya N, Sinha V, Ramachandra T, Mani S. Model-Driven approach for managing human interface design life cycle. MoDELS. 2007. 226–240.
 - [32] Nunes DA, Schwabe D. Rapid prototyping of web applications combining domain specific languages and model driven design. ICWE. 2006. 153–160.
 - [33] Freudenstein P, Nussbaumer M, Allerdig F, Gaedke M. A domain-specific language for the model-driven construction of advanced web-based dialogs. Proc. of the 17th International conference on World Wide Web. 2008. 1069–1070.
 - [34] Tongrungrrojana R, Lowe D. WIED: A Web Modelling Language for Modelling Architectural-Level Information Flows. J. Digit. Inf. 2004, 5(2).
 - [35] Tongrungrrojana R, Lowe D. WebML+: a Web modeling language for forming a bridge between business modeling and information modeling. SEKE. 2003. 17–24.
 - [36] Wan J, Bieber M. GHMI: A general hypertext data model supporting integration of hypertext and information systems. HICSS (2) 1996. 47.
 - [37] Wan J, Bieber M, Wang JTL, Ng PA. LHM: a logic-based hypertext data model for integrating hypertext and information systems. HICSS (3) 1995. 350.

- [38] O'Reilly T. What Is Web 2.0. O'Reilly Network. Available from: www.oreillyn-et.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html. Retrieved 2006-08-06.
- [39] Valverde F, Pastor O. Facing the technological challenges of Web 2.0: A RIA model-driven engineering approach. WISE 2009. 2009. 131–144.
- [40] Fons J, Valderas P, Ruiz M, Rojas G, Pastor O. OOWS: A method to develop Web applications from Web-oriented conceptual models. Proc. of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI). Orlando, FL – USA, July 2003.
- [41] UWE-UML based Web Engineering. <http://uwe.pst.ifi.lmu.de>, 2-3-2008.
- [42] Sakowicz B, Murlewski J, Labus A, Napieralski A. JWay - Model-driven J2EE application framework. Proc. of the International Conference of Mixed Design of Integrated Circuits and Systems. 2007. 703–706.
- [43] Cicchetti A, Di Ruscio D. Decoupling web application concerns through weaving operations. Science of Computer Programming, 2008, 70(1): 62–86.
- [44] Brambilla M, Fraternali P, Tisi M. A metamodel transformation framework for the migration of WebML models to MDA. In: Koch N, Houben GJ, Vallecillo A, eds. The 4th International Workshop on Model-Driven Web Engineering (MDWE 2008), CEUR Proceedings. 2008, 389: 91–105.
- [45] Moreno N, Fraternali P, Vallecillo A. A UML 2.0 profile for WebML modeling. Workshop Proc. of the Sixth International Conference on Web Engineering, Second International Workshop on Model Driven Web Engineering (MDWE'06). 2006.
- [46] Moreno N, Fraternali P, Vallecillo A. WebML modeling in UML. IET Software Journal, 2007.
- [47] Schauerhuber A, Wimmer M, Kapsammer E, Schwinger W, Retschitzegger W. Bridging WebML to model-driven engineering: from document type definitions to meta object facility. IET SOFTWARE, 2007, 1-3: 81–97.
- [48] Fatolahi A, Somé SS, Lethbridge TC. A meta-model for model-driven development of Web applications. Technical Report, July 2010. www.site.uottawa.ca/eng/school/publications/techrep/2010/TR-2010-04.pdf.
- [49] Dilmaj. http://sokhangozaar.appspot.com/?locale=en_US#, July 2010.
- [50] Fatolahi A. An integrated visual language for modeling Web UIs. modewis. blogspot.com/2010/08/integrated-visual-language-for-modeling.html.
- [51] OMG. MOF QVT Final Adopted Specification, November 2005.
- [52] Fatolahi A, Somé SS, Lethbridge TC. Automated generation of abstract Web models using QVT relations. September 2010, Technical Report TR-2010-06, School of Information Technology and Engineering, University of Ottawa. www.site.uottawa.ca/eng/school/publications/techrep/2010.
- [53] Fatolahi A, Somé SS, Lethbridge TC. Towards a semi-automated model-driven method for the generation of Web-based applications from use cases. Proc. of the 4th Model Driven Web Engineering Workshop (MDWE2008). MoDELS'2008. 2008. 31–45.
- [54] Fatolahi A, Somé SS, Lethbridge TC. A model-driven approach for the semi-automated generation of Web-based applications from requirements. Proc. of International Conference on Software Engineering and Knowledge Engineering. 2008. 619–624.
- [55] Mellor SJ, Scott K, UHL A, Weise D. MDA Distilled: Principles of Model-Driven Architecture. Boston: Addison-Wesley, c2004.
- [56] JDBC Overview. <http://java.sun.com/products/jdbc/overview.html>, July 2010.
- [57] Java Data Objects. <http://java.sun.com/jdo/>, July 2010.
- [58] Lethbridge TC, Laganière R. Object-Oriented Software Engineering: Practical Software Development using UML and Java. London : McGraw-Hill, 2001.
- [59] Stéphane Sotèg Somé's homepage. <http://www.site.uottawa.ca/~ssome/>, Fall 2008.
- [60] mediniQVT – Trac. <http://projects.ikv.de/qvt>, 3 May 2008.
- [61] EMF. <http://www.eclipse.org/modeling/emf/>, September 2009.
- [62] Yed graph Editor. http://www.yworks.com/en/products_yed_about.html, February 2010.
- [63] MODEWIS_PIM_Generator. September 2010, site.uottawa.ca/~afato092/modewis_pim_generator.jar.
- [64] MODEWIS_GWT_Generator. September 2010, site.uottawa.ca/~afato092/modewis_gwt_generator.jar.

- [65] MODEWIS_ProofRead. site.uottawa.ca/~afato092/modewis-proofread.vsd, September 2010.
- [66] Visio Information. visio.mvps.org, March 2010.
- [67] Cockburn A. Writing Effective Use Cases. Boston: Addison-Wesley, c2001.
- [68] Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Prentice Hall PTR. 2005.
- [69] Meliá S, Gómez J, Koch N. Improving Web design methods with architecture modeling. In: Bauknecht K, Pröll B, Werthner H, eds. The 6th International Conference on Electronic Commerce and Web Technologies (EC-Web 2005). Copenhagen, Denmark, LNCS 3590, Springer-Verlag. August 2005. 53–64.
- [70] The Standish Group International Report. Extreme Chaos, 2001.
- [71] Verner J, Cox K, Bleistein SJ, Cerpa N. Requirements engineering and software project success: an industrial survey in Australia and the US. Australian Journal of Information Systems, 2005, (13): 225–238.
- [72] Han WM, Huang SJ. An empirical analysis of risk components and performance on software projects. Journal of Systems and Software, 2007, 80(1): 42–50.
- [73] Molina F, Pardillo J, Ambrosio J, Álvarez T. Modelling Web-Based Systems Requirements Using WRM. WISE Workshops 2008. 122–131.
- [74] Escalona Cuaresma MJ, Aragón G. NDT. A model-driven approach for Web requirements. IEEE Trans. on Software Engineering, 2008, 34(3): 377–390.
- [75] Escalona MJ, Torres J, Mejías M, Reina AM. NDT-tool: A tool case to deal with requirements in Web information systems. Proc. of the Fourth Int'l Conf. Web Eng. 2003. 212–213.
- [76] Lee H, Lee C, Yoo C. A scenario-based object-oriented methodology for developing hypermedia information systems. 31st IEEE Annual Conference on Systems Science. Sprague R, 1998. 121–138.
- [77] Suh W, Lee H. A methodology for building content-oriented hypermedia systems. Journal of System Software, 2001, 56: 115–131.
- [78] Weidenhaupt K, Pohl K, Jake M, Haumer P. Scenarios in system development: current practice. IEEE Software, 1998, 2: 34–45.
- [79] Koch N, Zhang G, Escalona MJ. Model transformations from requirements to web system design. Proc. of the 6th International Conference on Web Engineering, 2006. 281–288.
- [80] Liang X, Kop C, Ginige A, Mayr HC. Turning concepts into reality - bridging requirements engineering and model-driven generation of Web-applications. In: Filipe J, Helfert M, Shishkov B, eds. Proc. of the Second International Conference on Software and Data Technologies (ICSOF 2007). INSTICC Press, Barcelona, Spain, 2007. 109–116.
- [81] Selic B. Model-driven development: its essence and opportunities. Ninth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC). 2006. 313–319.
- [82] Cicchetti A, Di Ruscio D. Decoupling Web application concerns through weaving operations. Science of Computer Programming, 2008, 70(1): 62–86.
- [83] Schmidt D, Stal M, Rohnert H, Buschmann F. Pattern-Oriented Software Architecture. Chichester [England], New York: Wiley, 2000-c2007.
- [84] CPPCMS. <http://cppcms.sourceforge.net/wikip/en/page/main>, December 2010.
- [85] Fusebox. <http://www.fusebox.org/>, December 2010.
- [86] ASP.Net. www.asp.net, December 2010.
- [87] ColdFusion. <http://cfwheels.org/>, December 2010.
- [88] Apache Struts. <http://struts.apache.org/>, December 2010.
- [89] JavaServer Faces Technologies. www.oracle.com/technetwork/java/javace/javaserverfaces-139869.html.
- [90] Seam Framework. <http://seamframework.org/>, December 2010.
- [91] Ruby on rails. www.rubyonrails.org, December 2010.
- [92] Aida/Web Smalltalk. <http://www.aidaweb.si/>, December 2010.
- [93] Catalyst Web Framework. <http://www.aidaweb.si/>, December 2010.
- [94] AMS_APSM. http://www.site.uottawa.ca/~afato092/AMS_APSM_Ali_Fatolahi.zip, December 2010.
- [95] MODEWIS_yEd_Workshop. http://www.site.uottawa.ca/~afato092/modewis_yed_workshop.zip,

Dec 2010.

- [96] Sokhangozaar. <http://code.google.com/p/sokhangozaar/>, January 2010.
- [97] Lowe D, Henderson-Sellers B, Gu A. Web Extensions to UML: Using the MVC Triad. ER 2002. 105–119.
- [98] Gu A, Lowe D, Henderson-Sellers B. Web modelling languages: the gap between requirements and current exemplars. Proc. of Australian World Wide Web Conference 2002. <http://ausweb.scu.edu.au/aw02/papers/refereed/lowe/paper.html>.
- [99] Meta-Model Discussions. Official WebRatio Forum Notes. groups.google.com/group/webratio/browse_thread/thread/1d5fe8425ea77da1?hl=en. 15 April 2011.
- [100] Almeida PJ, Dijkman R, van Sinderen M, Pires LF. On the notion of abstract platform in MDA development. Proc. of the 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004). 253–263.

Appendix A – The Botterweck Model

Botterweck’s model includes the following packages - More details are provided in Ref.[48] -

- State Machine includes the elements required to build a state machine in accordance with UML state machines.
- UI Structure encloses the elements required to build the general structure of the UI model such as pages, units and navigation links.
- UI Components includes the modeling elements for the UI components used for communications with end users.
- Data Model follows UML’s core model for specifying classes, objects, data types and their attributes.
- Data Components is a part of the model that relates the data model to UI components in order to transfer data in and out of the presentation layer.
- Web Services is a part of the model that is provided to give support to web service applications. The package can be used for regular web applications as well.

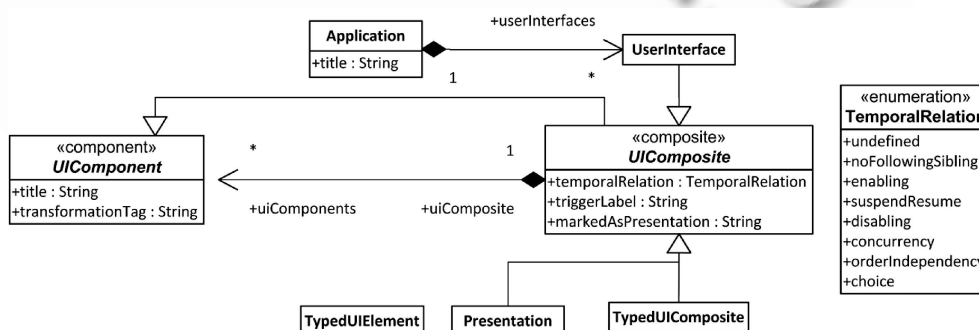


Figure A1. UI structure model from Ref. [17]

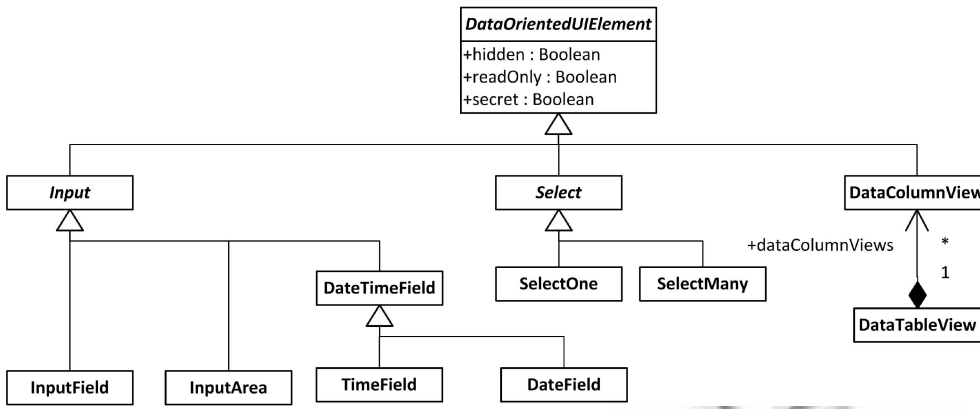


Figure A2. Data-oriented elements of UI components from Ref. [17]

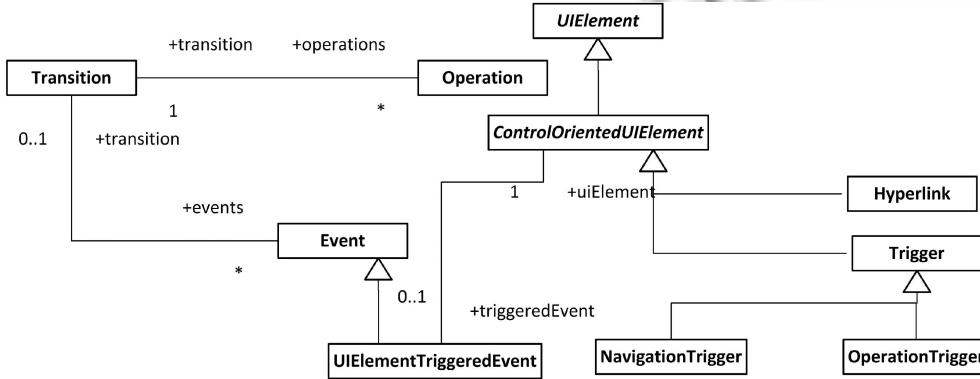


Figure A3. Control-oriented elements of UI components from Ref. [17]

Figure A1 details the UI Structure package. According to this model, an application can have several user interfaces. A user interface is a special kind of UI composite, where a UI composite is a collection of UI components. UI components are elements that the user can see and directly communicate with. A UI composite may be a presentation, which in association with a state makes a presentation state meaningful. This package also contains the abstract elements used by concrete UI elements in a UI element model.

Figure A2 shows a subsection of the UI components model. These components include: components connected to data elements, and composite elements such as input fields, selection boxes and table views. Figure A3 presents the model for plain hypertext, links and buttons. Triggers occurring in a presentation could be of one of the following three types: hyperlinks, operation triggers such as submit buttons that require an action in the controller and/or service layer and navigation trigger that only cause a change in navigation path.

The following is a list of important associations that exist on other parts of the abstract model that are required for better understanding of mappings and transformations:

- A transition can be associated to several Operations. This capability is used to model controller operations (Fig. A4).

- Transitions can also be associated to several Events. This serves to model signal events over the transitions. UI triggered events are a special type of events, which can have operation triggers as triggers (Fig. A4).

- Data Oriented elements are associated with Data Composites that are collections of data elements. Data composites are connected with a special class called OperationAdapter, which could be assigned one of the following roles corresponding to CRUD operations (Fig. A5).

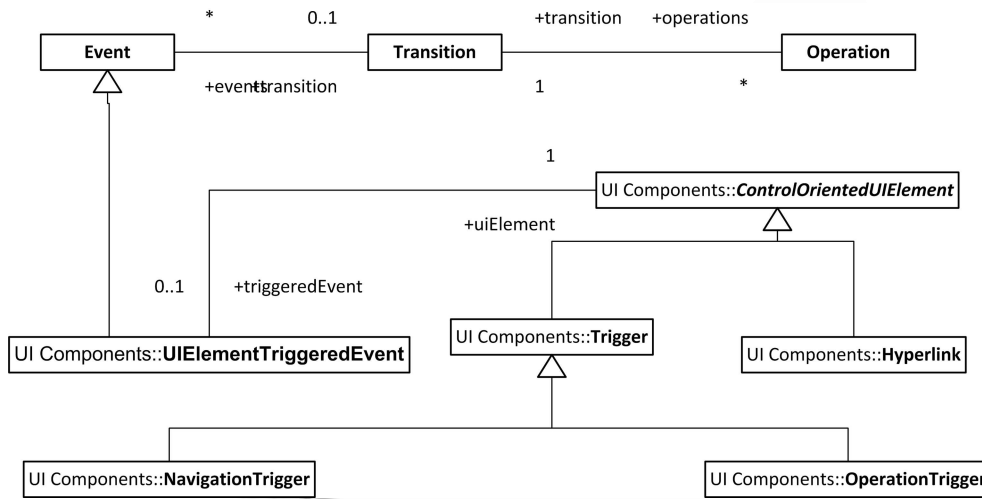


Figure A4. Transition in relation with events and operations from Ref. [17]

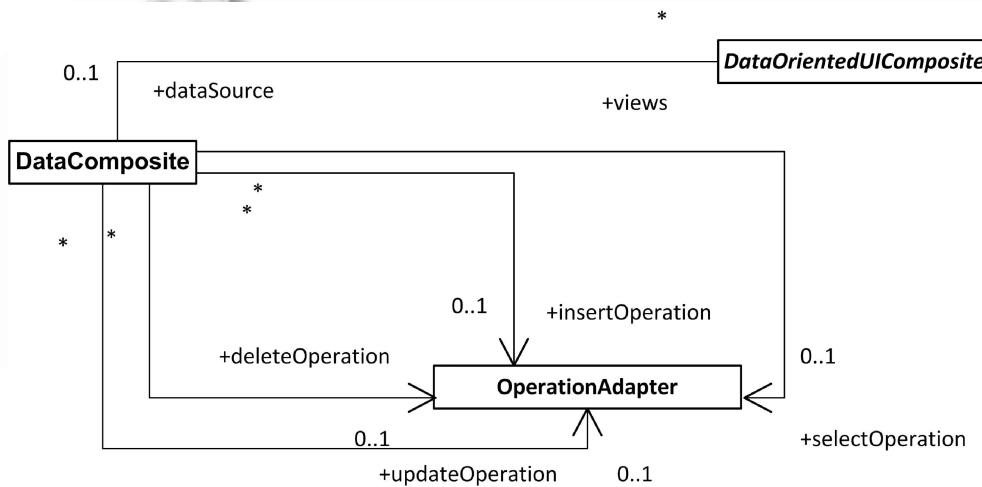


Figure A5. Data composite related to operation adapters Ref. [17]