A Meta-Model for Model-Driven Development of Web Applications

Ali Fatolahi Stéphane S. Somé Timothy C. Lethbridge

afato092@site.uottawa.ca ssome@site.uottawa.ca tcl@site.uottawa.ca

School of Information Technology and Engineering

University of Ottawa

Abstract. *Model-driven development of web applications has acquired much attention in recent years. Model-driven techniques, often transform a high-level input model to lower level models or executable code. This calls for the transformation rules that bridge source and target models. Those models are typically defined based on meta-model(s). A number of abstract models have been proposed for web applications in the past. In this report, we present a meta-model for web applications based on an existing model. We argue that our meta-model is appropriate to be used at an abstract level and could be mapped to multiple platforms. We will also discuss specific features of our meta-model, which affirm its handiness for model-driven development.*

Keywords: Web, Meta-Model, MDD, UML.

1. Introduction

Model-driven techniques have been used in the area of web development to a great extent. Such methods require precise definition of the meta-models used as the source and the target of the 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 technological volatility of the World Wide Web. In this report, we present a UML-based meta-model of web applications to be used in the context of model-driven methods targeting several platforms.

The ability of a model-driven method to operate on different platforms depends on the designed meta-models as well as the complexity and flexibility of the transformations that map the abstract models to specific platforms. Many of the model-driven methods of web engineering are tuned towards specific platforms or develop only certain parts of web applications. However, if a method aims at supporting multiple web platforms, it would need a meta-model that describes all features of web applications at an abstract level. The suitability of such abstract models may be validated by transforming them to different platforms. In this document, we argue that our meta-model could be mapped to a other specific web platforms by presenting example mappings from our meta-model to specific platforms such as AndroMDA (2007), WebRatio (2008) and Google Web Toolkit (GWT, 2010).

Many model-driven web engineering methods aim at using UML-based models. The preference of UML as a popular modeling language gives the designers the ability of

building their modeling languages based on a well-formed language that is supported by the common knowledge of many researchers and developers. This will make the new language easily understandable and applicable. The most applied models in the area of web engineering such as WebML (2008) and UWE (2009) are in fact Domain Specific Languages (DSL). However due to the popularity of UML, there has even been efforts to bridge those DSLs with UML. The model presented in this document is an extension to a UML-based language for specifying web applications according to UML 2.1.1 (OMG, 2007).

The meta-model we develop in this report, 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 (OMG, 2003); we name this type of PSM, Abstract PSM (APSM). Therefore, the APSM is defined upon an abstract model for web-based applications. The APSM is a set of models delimiting universal necessities of web-based applications regardless of the platform on which the application is deployed. PSMs built specifically based on concrete platforms are the results of transforming APSMs and are called SPSMs. We will demonstrate examples of mapping this APSM to models of specific platforms i.e. PSMs.

One of our goals in distinguishing these two levels of PSM is to facilitate the expression of transformations from platform-independent models (PIMs) to platform specific models. According to Jinkui et al. (2007), a large gap between a source model and a target model introduces difficulties when devising mapping rules for model-driven methods; such hardships refer to the large distance from PIM to PSM that necessitates more complicated and hence error-prone mappings. Breaking the transformation up two lighter transformations would shorten the gap and would facilitate designing model-driven transformations. By relieving developers from low-level platform specific related design, our meta-model has the potential to shift the development task to issues related to business needs. Another benefit is the abridged development time. This could help web developers to overcome the problem of schedule delays, which is recognized as one of the top five most-cited problems with large-scale web systems mentioned by Epner (2000).

The meta-model has been designed independently from the physical aspects of the database. Such a distinction proves useful in many situations. For example, many applications require data migration or reformatting after a while. In such cases, the changes required to the database could happen without the need to change the application models. The assumption of a pre-existing database is also realistic as many practical projects that are defined for upgrading the legacy software applications to a web-based version whiles the database are kept intact.

The rest of this document is organized as follows. In Section 2, we present a survey of the existing models of web applications. In Section 3, we explain the details of a selected existing model. Section 4, contains our meta-model that extends the model of Section 3.

Section 5 proposes a model-driven method to work with our meta-model. Section 6 details an example and Section 7 concludes this paper.

2. Models of Web Applications

Several models for specifying web information systems exist. The study, categorization and selection of models start with a few general assumptions. The assumptions are:

- The model must capture the abstract structure and behavior of a web-based application
- an abstract UI model should be included
- the abstract data model should exist and be connected to the UI model and the model of behavior

We have studied a number of models proposed for both web-based applications and UI modeling with an emphasis on data access design. The existing models and methods are categorized in four different groups. First, we distinguish models defined to describe a web application model as well as its UI model. The second group involves the models of abstract web applications. The third set of models consists of abstract UI models. The fourth group belongs to the works studying parts of the application UI such as its dialogs. Finally, other work addressing code generation methods using UI models are considered.

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 (2000). WebML provides a notation to define the UI presentation layer and the semantics to specify web-based applications in a multi-layer approach. In recent years, there has been a number of efforts to relate WebML with the UML/MOF family (OMG, 2002), e.g. Moreno et al. (2006), Moreno et al. (2007), Schauerhuber et al. (2007) and Brambilla et al. (2008) that witness the popularity of WebML.

As another example, UWE (Kroiß and Koch, 2008) 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, 2009) for web development. A series of related work in this area is dedicated to UWE. 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.

Botterweck (2007) 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. 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. This model is discussed in more details in Section 3.

Muller et al. (2003) provide another abstract model with the same intentions as Botterweck's but based on a different approach. Muller et al.'s work is similar to ours in terms of breaking up the PSM level into two levels: One is named platform-dependent PSM and another would be 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 (2005) 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. (2006), which provides a multi-layer architecture for web-based applications in alliance with the Model-View-Controller (MVC) pattern (Schmidt et al 2007). He et al. (2005) report an interesting approach to define a role-based abstract model for web-based applications. However, the method is rather a general method for creating abstract web models.

Another group of existing models are those that address abstract UI models with less connection to web applications. Blankelhorn (2004) has provided a UML profile for GUI layouts. Da Silva and Paton (2003) 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 (2005) presents an MDA-compliant environment for designing user interface models of information systems in general. Schattkowsky and Lohmann (2006) introduce a general method for designing platform-independent UIs but the method is not supported by any specific meta-model. Diamodl (2008) 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 (2000) illustrate a how-to UML-based approach to devise the architecture of web-based applications in a three-tier framework. Kavaldjian (2007) and Bogdan et al. (2008) describe a method to extract a UI model from a discourse model in a general sense. Costa et al. (2007) present an MDA-compliant method to devise a step in user interface design for web applications in accordance with UML version of ConcurTaskTree (CTT). The new language is called Canonical Abstract Prototype 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 in order to reach 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 PHP's application Model2Code. De Souza et al. (2008) have published

another model-driven approach to generate web UI. 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 UI design model, which is followed by the implementation of UI. An eclipse plug-in is implemented to hard-code the transformation rules required to generate JSF code. Sukaviriya et al. (2007) introduce a process framework for model-driven UI design based on iterative/incremental approaches. Arraes Nunes and Shwabe (2006) 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 the conceptual instances plus navigation model and abstract UI to produce HTML pages.

The related work studying existing models addressing parts of the web UI form another group of the literature review. As an example, Freudenstein et al. (2008) 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. Tongrungrojana and Lowe (2004) 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 Tongrungrojana and Lowe's previous work to present *WebML+* (2003), a language for modeling information exchange in the workflow models. Some older approaches such as LHM introduced by Wan et al. (1995) and GHMI by Wan and Bieber (1996) exist that could be counted as early efforts to integrate classic information systems with hypertext features. Another similar approach is taken by Whitehead et al. (1994) to describe an approach to bring repositories to the hypertext level so that they could be visualized using the web.

Table 1 renders a scoring mechanism for existing models, which justifies our selection of the model. In this table, only the most related work of the existing work discussed above are contained. Following is a description of the features mentioned in this table:

- Abstract Web: Is the model able to define an abstract web application? A model may pass this criteria if it covers basic components of a general web application including the navigational aspects and contents of the web, UI model and data access mechanisms. The model also needs to not be dependent on specific technologies.
- Abstract UI Model: Does the model have the capabilities of specifying abstract UI models?
- UML Compliant: Is the model compliant with UML in terms of supporting the same kind of modeling elements or extending the existing UML elements?
- Separation of Concerns: Separation of concerns plays a key role in designing software applications by increasing flexibility and reusability as well as keeping

different parts of the design independent. This feature has been approached using a variety of techniques including the MVC pattern.

- Coverage to UI Components: It is important to have the details of UI components modeled. This includes a precise categorization of UI components and their relationship to data and operations. Several types of UI components exist on web applications. Therefore, modeling a web application without details of each category would not be possible.
- Details of the UI Model Connected to other Components: Does the model elaborate the UI/Web model in relation with other parts of the application such as the behavior and data? Such connections are required either to delineate the automated generation of the application based on the input model or to guarantee the conformance of the resultant model to the MVC.
- Support to data-related operations: Many of the models such as WebML support data modeling and data access. However, this factor means if the abstract model provides concrete modeling elements that represent specific CRUD operations?
- Availability of a detailed Meta-Model: This is perhaps the most important factor and the key to our selection. In order to define detailed mappings and transformations, we need access to the entire meta-models with details of elements and their relationships. We have sought meta-models in two different ways: one was to look for different publications addressing each model and the second way was to approach authors for more details.
- Requirements as Input: Formal incorporation of requirements from the beginning of the development process can help the quality of the application as well as steering the development process in the direction of requirements. It is not possible to formally include all types of requirements in the development process. The desired UI as one type of usability requirements form an instance of requirements that could be formally included in the process by defining the desired UI model as an input.
- Security Features: Matters of authentication and securing web applications at an abstract level form another important factor. Implementation of such features may differ for specific platforms but an abstract level would be devised to indicate whether access to special units requires authentication or not and if the contents of some units must be secured.
- Web-Independence: We discussed the importance of positioning the meta-model at an abstract level as opposed to specific web platforms. We also mentioned that our meta-model may be called an APSM since it is dependent to an abstract specification of web i.e. abstract web platform. It is, also important to note that every model-driven method requires an input platform-independent model (PIM). Considering abstract web as the platform, the PIM would be web-independent. Method designers may choose to have two different meta-models for PIM and APSM but it would be a great advantage of a meta-model id a subset of it could be used for specifying the PIM. A few models provided for specifying web applications are specifically designed for web applications but a number of other models could be used in whole or partially to specify other applications as well. We prefer the usage of former ones.

As Table 1 shows, Botterweck's model scores higher, even though it is rather a general UI model than a concrete web application model. This does not cause a major issue since the model has enough modeling elements to support an abstract web application as well. Another advantage of this model over the more famous web modeling languages such WebML and UWE is that although there has been efforts to comply them with UML, they are not originally UML-based according to Brambilla et al. (2008) while Botterweck's model is a direct extension to UML, which we find it more appropriate because of the popularity of UML. It is also worth mentioning that Botterweck's model provides support for advanced UI elements such as multimedia contents, which is not present in most other languages. Table 1 also draws attention to the fact that none of the studied models prime the entry of requirements as an input along with models of UI, data and expected operations.

Table 1 – Comparative Study of Existing Abstract Models

	UWE	W2000	WebML	Botterweck's	Muller et al's	Nikolaidou and Anagnostopoulos's	He et al's	Blankelhorn's	UMLi	Vanderdonckt	Schattkowsky and Lohmann	Diamodl
Abstract UI Model	*		*	*	*	*		*	*	*	*	*
Abstract Web	*	*	*	*	*		*					
Availability of Meta-Model in Details				*								
Coverage of UI Components	*		*	*	*			*	*			
Details of the UI Model Connected to other Components			*	*								
MVC Compliant	*	*	*	*	*	*	*					
Support to data-related operations			*	*								
UML Compliant				*	*	*		*	*	*		
Requirements as Input												
Security Features			*									
Web-Independence				*	*			*	*	*		*

We can conclude that WebML-based approaches present the most comprehensive way of data modeling for web applications. Further, WebRatio (2008) as 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 databases. While WebRatio has integration with the database, our approach does not take the database into consideration. It is assumed that the database already exists and the target platform will handle the database. Therefore, the core of our method, which is the PIM-APSM transformation is not dependent on the database. It is also worth remembering that traditional web modeling languages including WebML are not based on UML/MOF according to Brambilla et al. (2008), even though there has been a number of efforts to adapt WebML with MDA/MOF/UML family such as the ones by Moreno et al. (2006), Moreno et al. (2007), Schauerhuber et al. (2007) and Brambilla et al. (2008). Another difference is that data modeling in WebRatio as the most famous WebML tool is largely manual while our approach automatically discovers the data model. Finally, WebML is specifically designed for web applications that is, it may not be used for specifying applications at a web-independent level while our meta-model consist in a web-independent subset that could be used for the PIM level specifications.

3. Botterweck's Model

In this section, we introduce Botterweck's model for web-based applications. As mentioned earlier, the model is used for both the source and target of transformations. This model is based on UML but as described in Section 5 we will extend it following UML 2.1.1. This model includes the following packages:

- *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 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. It allows some functionality to be introduced as services. The developer can then choose to deploy them as real web services or to implement as internal services. In the latter case, because the functionalities are tagged or stereotyped as services, it is always possible to eventually change them to web services.

Figure 1 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 2 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 3 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.

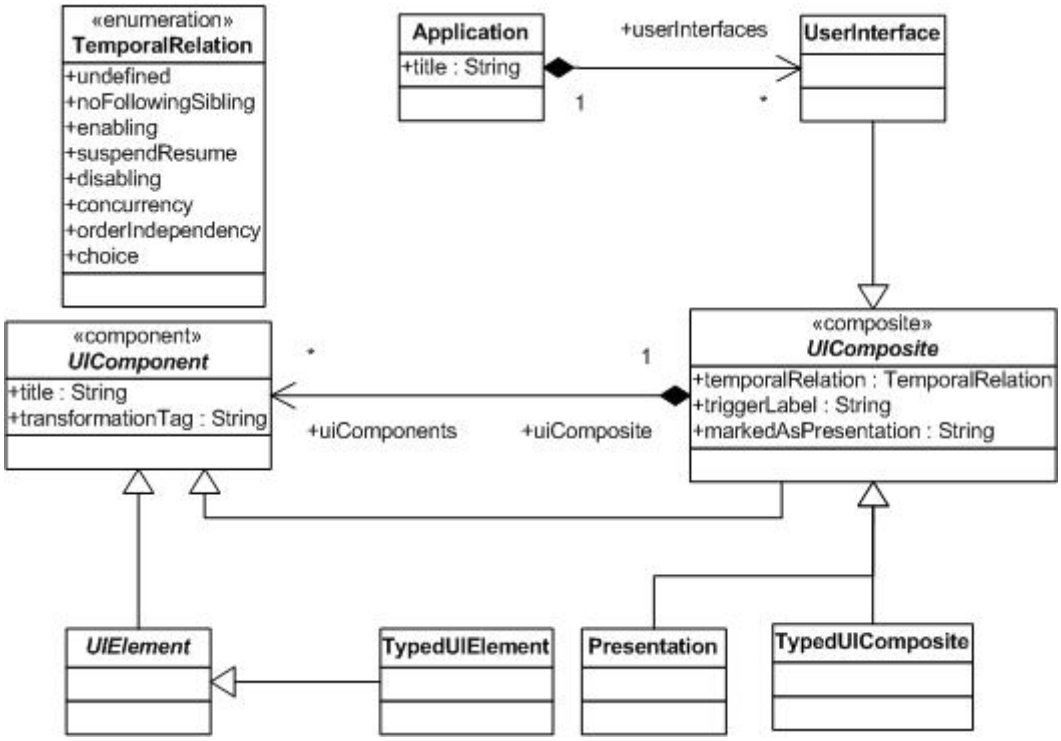

Figure 1 - UI Structure Model (Botterweck, 2007)

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 with several *Operation*s. This capability is used to model controller operations (Figure 4).
- Transitions can also be associated with 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 (Figure 4).
- 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 (Figure 5):
  - selectOperation
  - insertOperation
  - deleteOperation
  - updateOperation

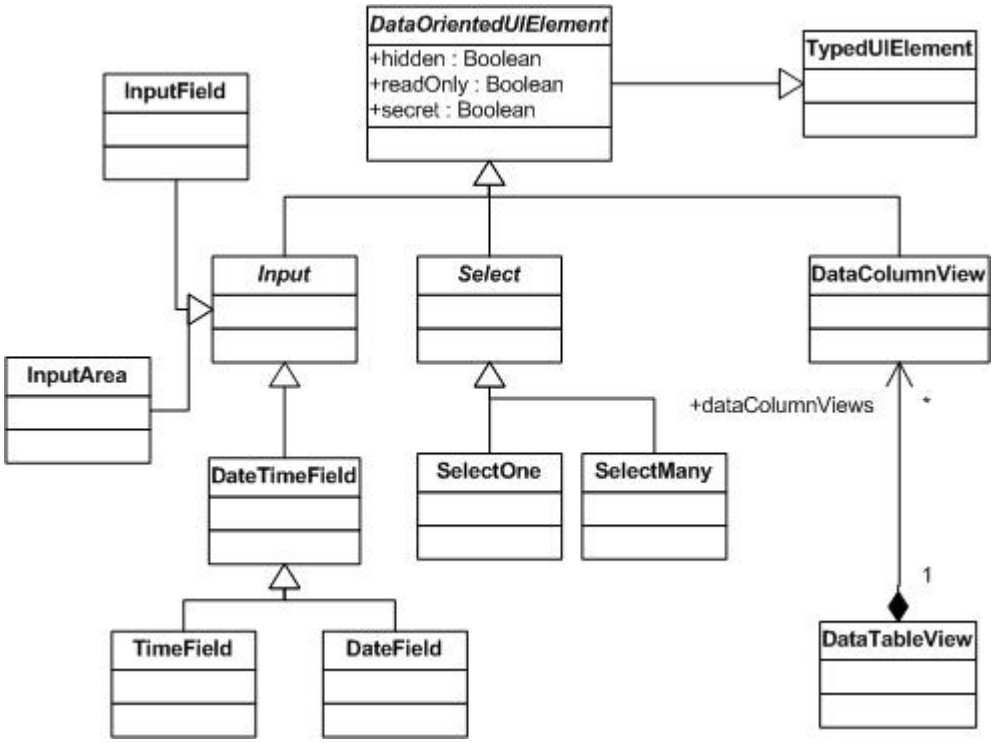

Figure 2 - Data-oriented elements of UI components (Botterweck, 2007)

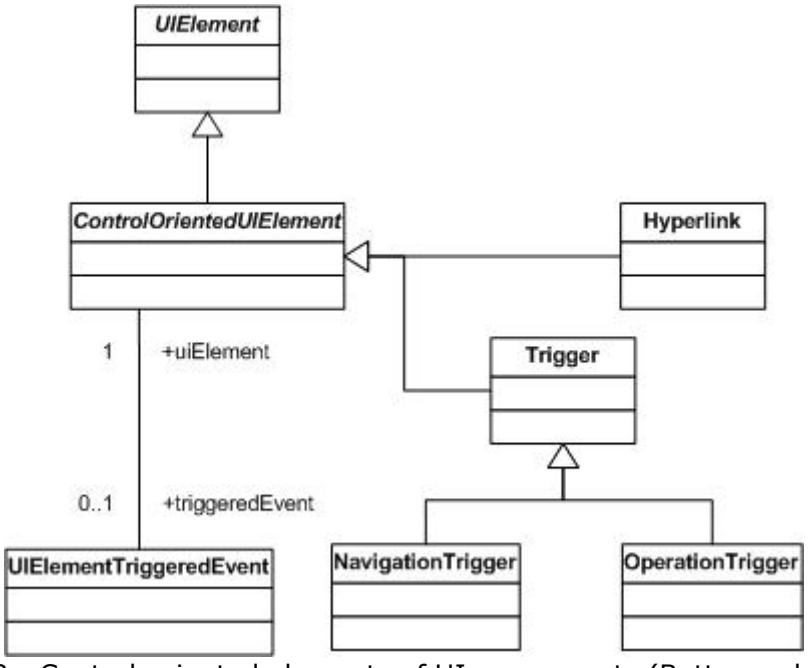

Figure 3 - Control-oriented elements of UI components (Botterweck, 2007)

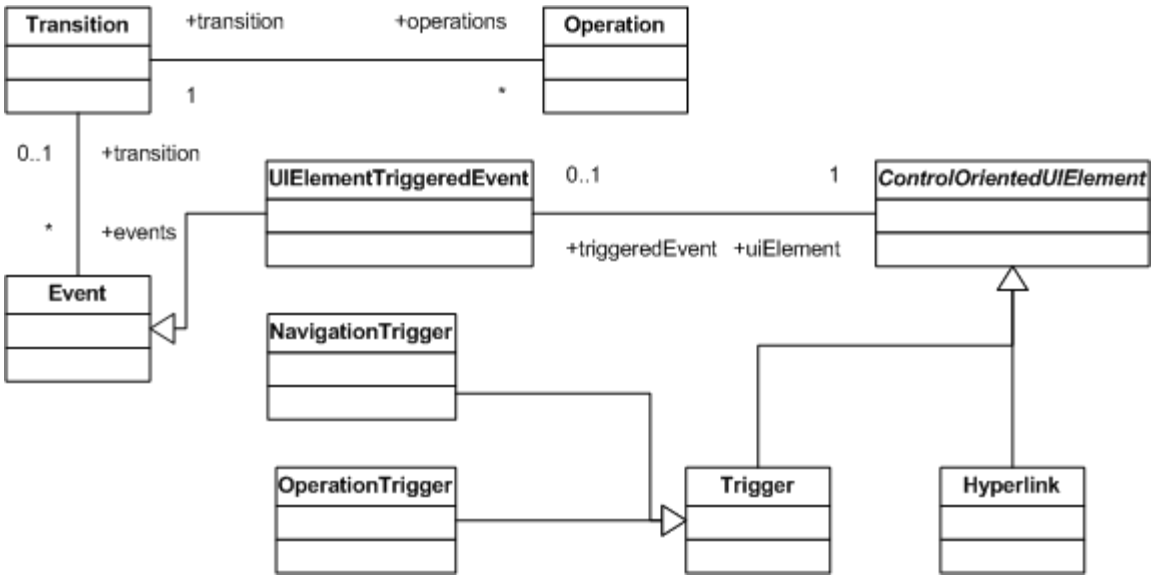

Figure 4 - Transition in relation with events and operations (Botterweck, 2007)

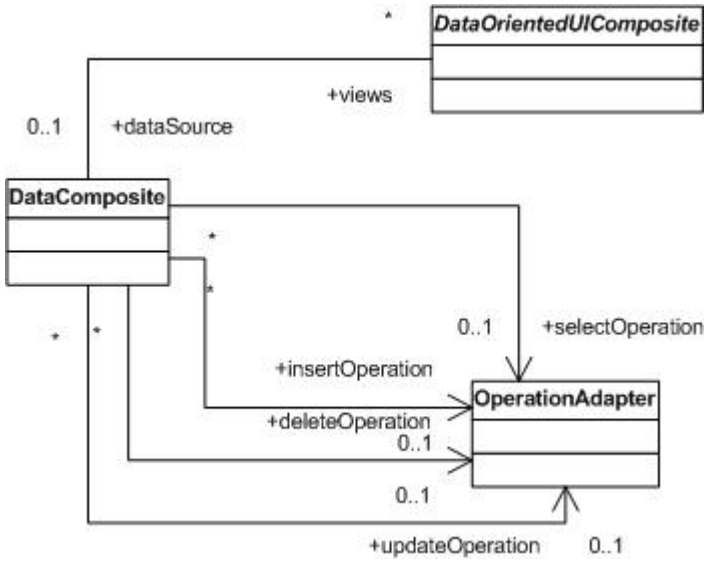

Figure 5 - Data Composite related to Operation Adapters (Botterweck, 2007)

4. Customizing the Abstract Model

The model described in Section 4 provides only a core model as the starting point towards a complete model of an abstract web application. This model is specified at a high level and hence at a far distance from specific web platforms. Several features such as security features, navigation modeling, details of data processing and presentation as well as the relationships of different layers of web applications are not seen. In fact, the chosen model is more suitable to be used at the PIM level. As a result, several

adjustments to support the essentials of describing the models at the level of APSM must be undertaken. Every of these changes are required due to one or more of the followings:

- The specific format of the input model, that is the approach taken to accept the user's input specifying the web application
- The format of the output model specially at the abstract level, where there is a need for an abstract model that is portable to virtually any web platform – that is the abstract models leads to models of web applications that carry the required details of transforming the APSM to specific PSMs.
- The details added to the either of either the input or the output models in order to make the automation possible.

Certain parts of Botterweck's model remain intact. These are mainly the elements that build the contents of the web application such as the input or multimedia elements. Other parts require changes. This is because according to Table 1, Botterweck's model does not support two of the required criteria. Also it supplies only partial support to some other criteria mainly because it aims at being a rather general model than a specific web application model. 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 on shop, pictures, feedbacks 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.

As a working example, consider Figure 6. In this figure, the selection of an element from the drop-down select box affects the contents of the list box that includes the list of students within the selected team. The model also asserts that the action of the button, *Remove* necessitates the removal of a member from a team. As Figure 6 shows, UI components may be attached to data elements. As mentioned earlier, it is assumed that a database exists or the design of the database is performed separately.

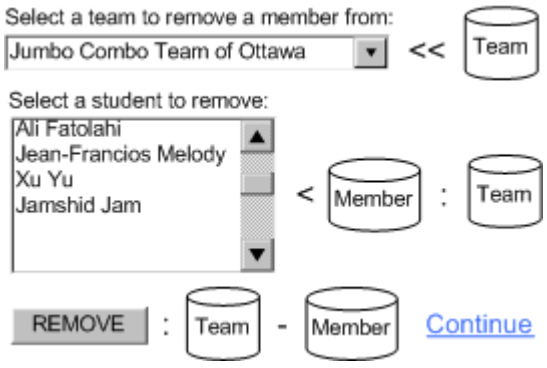

Figure 6 – A sample UI model

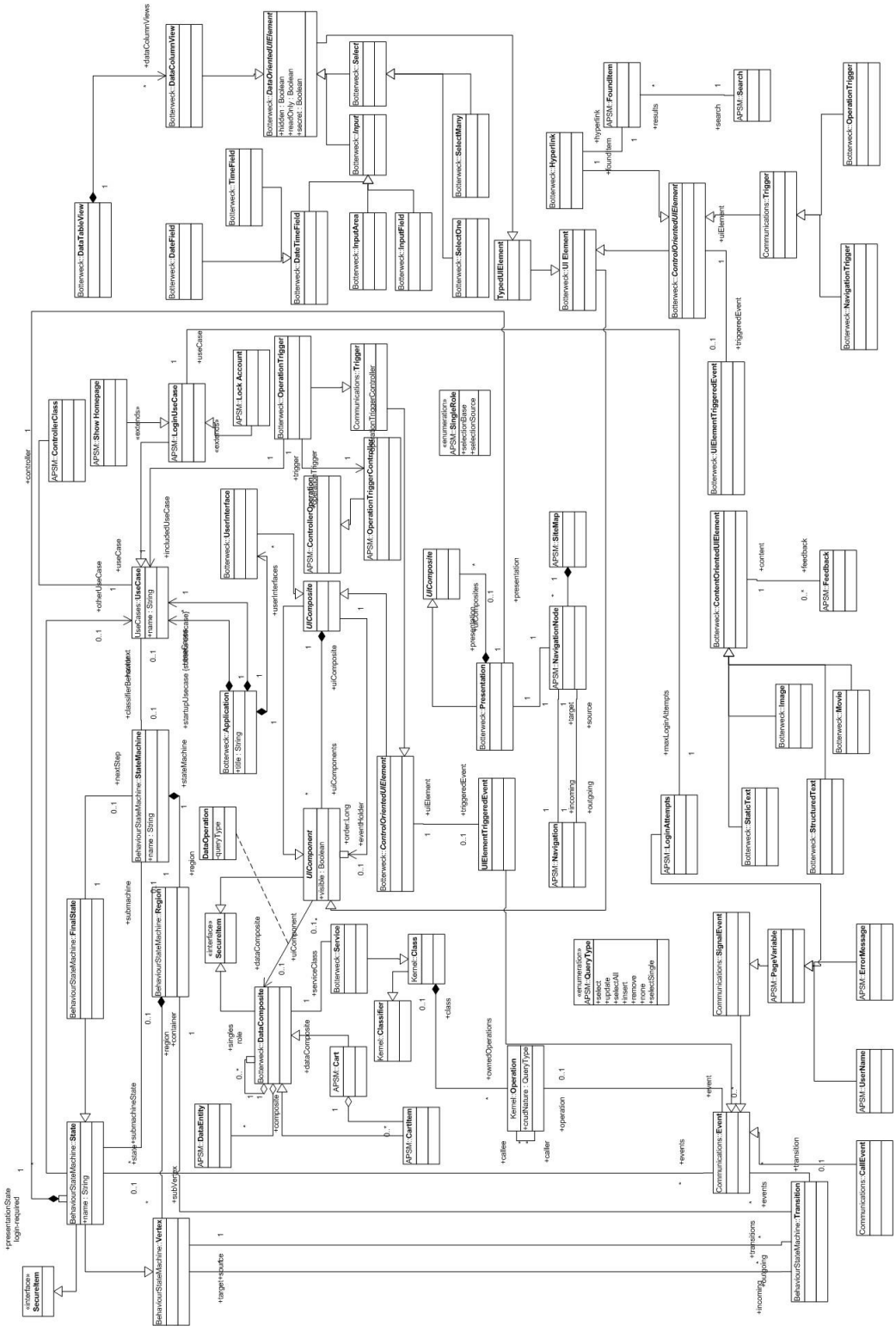

Figure 7 – A new meta-model based on Botterweck's model

Figure 7 presents the meta-model we will delineate in further subsections. An application is described as a number of use cases. One of the use cases is identified as

startup use case; that is, a user interaction with the system begins at that use case. Every use case is modeled as a state machine. We distinguish presentation states that correspond to points where user input is solicited or where output is provided. A presentation state may be specified as login-required, which is indicated as an association attribute in the model. Furthermore, login-required states may demand a check on a maximum number of login attempts. Each presentation state is associated to a presentation. A presentation has UI composites, which are composed of UI components. A *UIComponent* can be associated with a *DataComposite* that represents a data object or a composition of data objects. The association has a *DataOperation* class representing the type of association as one of the CRUD operations defined in *QueryType* enumeration. *OperationTriggers* represent input events placed on web entry units such as submit buttons. Operation triggers are also considered *UIComposites* so that they can carry UIComponents as entry fields. Finally, Transitions and States are allowed to accept events for control, logic or data related tasks. An event may call operations making it a call event.

Compared to the Botterweck's model, the model of Figure 7 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. In the rest of this section, these modifications are reviewed and formalized in more details.

4.1. 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 is 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. This model is presented in Figure 8. According to this figure, 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.

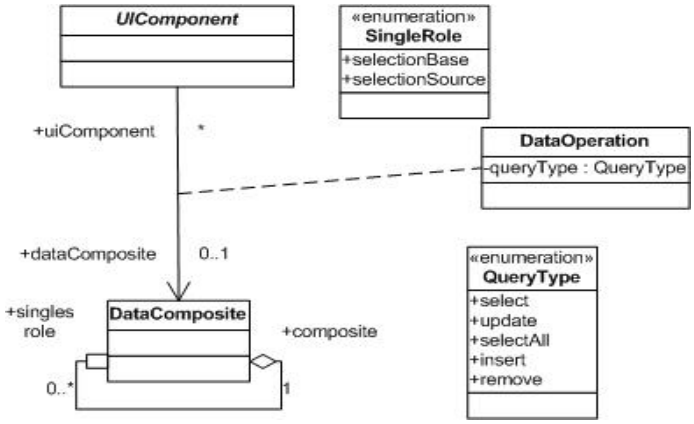

Figure 8 – UI Components supported by Data Composites

DataComposite could also be identified in a more complex way. For example, the statement *DataComposite1: DataComposite2* (such as the statement *Member: Team* in Figure 6) expresses the fact that a select component must be populated with data items from *DataComposite1* based on a previous selection of an item from *DataComposite2*. In such cases there must exist a *SelectOne* component within the same presentation, where *SelectOne* is associated with *DataComposite2*. This rule is formally defined using the following constraint:

```
context UIComponent
        inv:
                if self.dataComposite.singles->exists(dc|dc.role=SingleRole.selectionSource) then
                        self.presesetnation->exists(uic|uic.dataComposite=self.dataComposite)
                endif
```

In Figure 8, the association from *DataComposite* to itself supports associations with data composites composed of more than one data item. The association has two ends. The shared end is named *composite* and the end named *singles* carries the data composites that join together to build the main data composite. The association has an attribute on the *singles* side indicating the role of the data composite in the joint data. For example in case of Figure 6, the data composite *Team* takes the role *selectionBase,* while the data composite *Member* accepts the role *selectionSource* contributing to the data composite required to supply the list box of students.

Although data composites may be associated with any UI component, the support to data operations is limited to certain types of UI components such as operation triggers and data oriented UI elements and composites. The following OCL expression states this fact:

```
context DataOperation
        inv: self.uiComponent.oclIsKindOf(DataOrientedUIElement) or
             self.uiComponent.oclIsKindOf(DataOrientedUIComposite) or
             self.uiComponent.oclIsKindOf(ControlOrientedUIElement)
```

Note that *oclIsKindOf* is used instead of *oclIsTypeOf* because we need to check the type of the element not only against itself and but also its parents.

Other constraints apply as well. For example, a select component can only accept a query of *select* or *selectAll*:

```
context Select
        inv: self.DataOperation.queryType=QueryType::select or
             self.DataOperation.queryType=QueryType::selectAll
```

One advantage of the new association query to the original version is the fact that it chips in more freedom when defining new types of queries. This is critical for future expansions of the method, when more complex data queries are to be automatically generated. Also, with this type of data operations, one does not have to define a specific type of operation at the time of associating a data composite to a UI component. This could be postponed to a later time when the exact nature of the association has been determined.

4.2. Operation Trigger

As mentioned in Section 4.1, operation 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 Figure 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. Figure 9 shows the meta-model support of this fact.

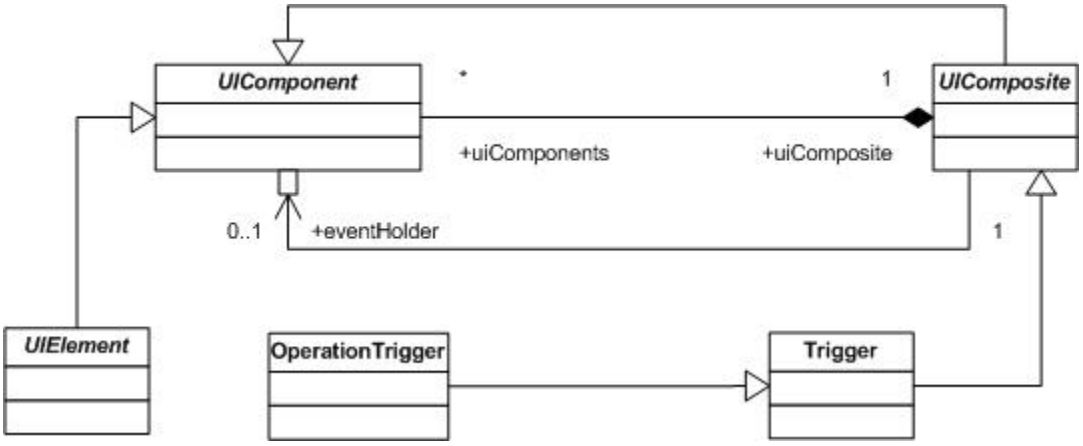

Figure 9 – Operation Trigger as a UI Composite

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 default event 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. This association is shown in Figure 10.

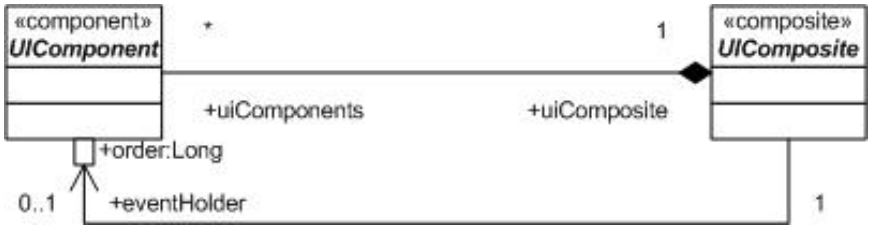

Figure 10 – UI Components may hold default event within a UI Composite

As Figure 10 shows, 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.

4.3. 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. Figure 11 shows the meta-model support for such cases. 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. The figure also shows that an operation trigger may accept a reference to another use case as a denotation of an included use case. This is called the *includedUseCase*.

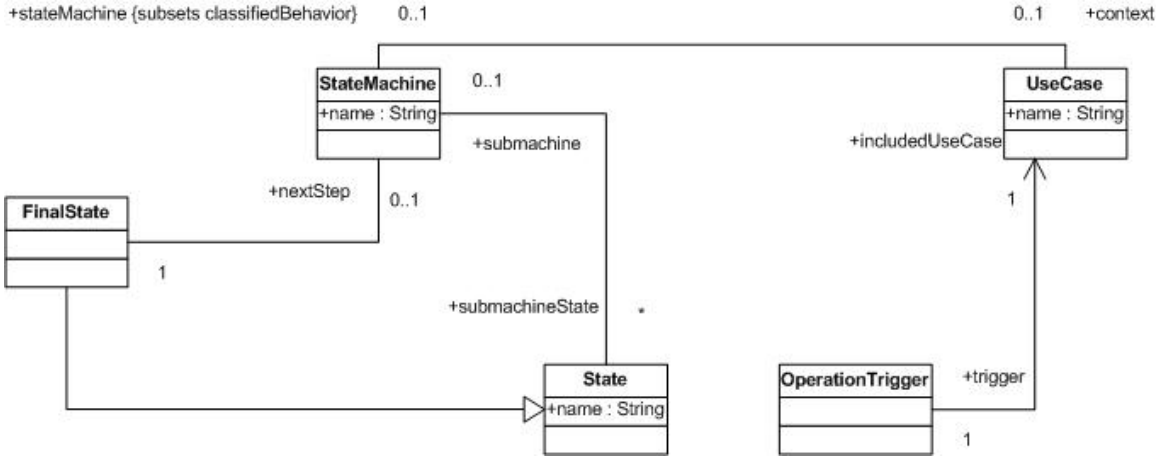

Figure 11 – Meta-Model support for Referencing State Machines from States

The following constraint applies to the model of Figure 11. The constraint does not limit final states because final states might be connected to a use case or not. In case they are, the connection does not imply any formal relationship between use cases but an informal flow of execution from one use case to another. The constraint is rather about the normal states that are associated with a use case other than the one being implemented by their holding state machine. In such cases, the use case is either included or an extension of the current use case.

```
context State
        inv:
                not self.oclIsTypeOf(FinalState) and
                if app.stateMachine.useCase.application.useCases->exists(uc | uc.name=self.name) then
                        uc.name=self.stateMachine.usecase.extend.extension.name or
                        uc.name= self.stateMachine.usecase.include.addition.name
                endif
```

The sequencing of state machines is indicated by the attachment of the final state to the use case corresponding to the state machine activated afterwards; 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. A more formal specification of sequencing state machines is as follows:

State machine *sm2* representing use case *uc2* should be sequenced after the state machine *sm1* representing use case *uc1*, if
1. successful execution of *sm1* i.e. validation of the postconditions of *uc1*, validates the preconditions of *uc2* or
2. *uc2* steps after *uc1* in the description of an arbitrary summary-level use case; summary-level use cases are use cases that are used to describe high-level scenarios, which may be composed of more than one use case according to Cockburn (2001).

4.4. Login Procedure

A default login state machine is primed in our model, which is shown in Figure 12. Figure 13 presents the meta-model fragment required to supply such a state machine. As Figure 13 shows, the state machine *Login* may lead to either state machine *Lock Account* or *Show Homepage* in case of failure or success respectively. It is also affirmed by a maximum number of failed login attempts.

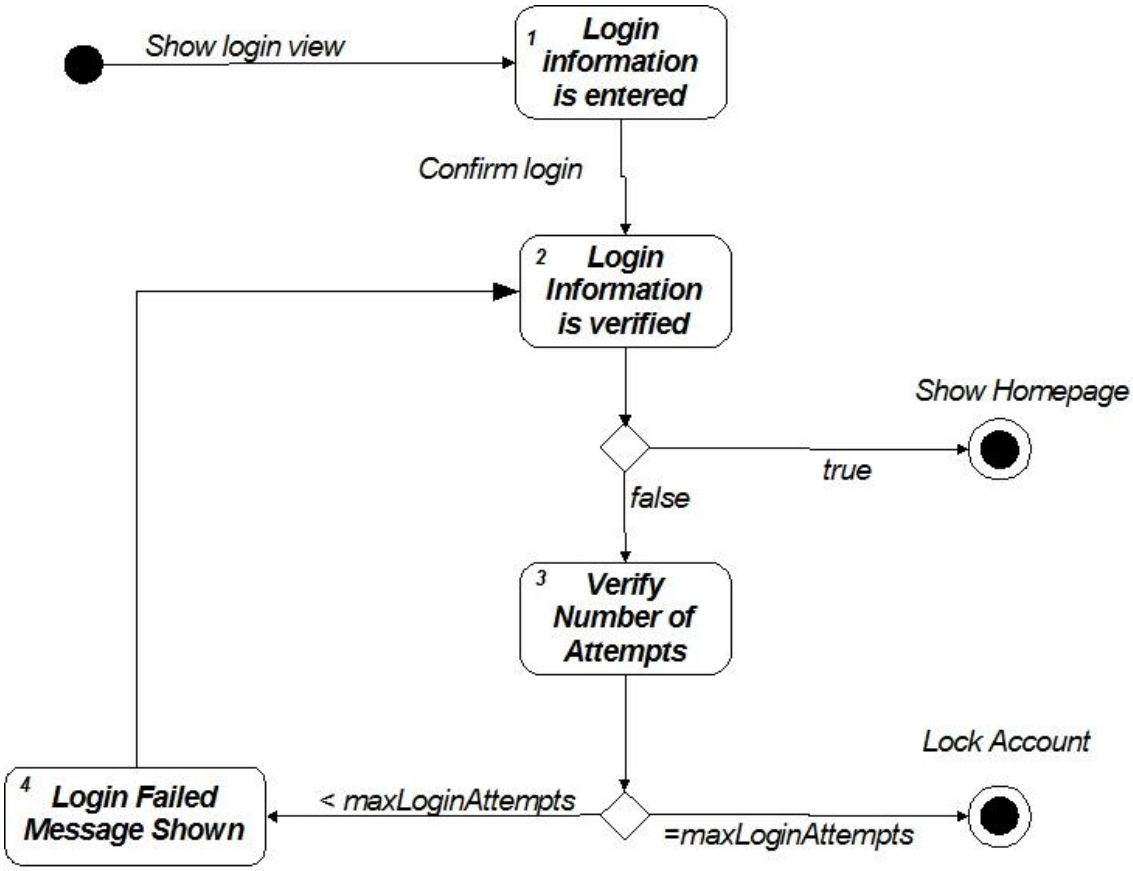

Figure 12 - An abstract Login State Machine

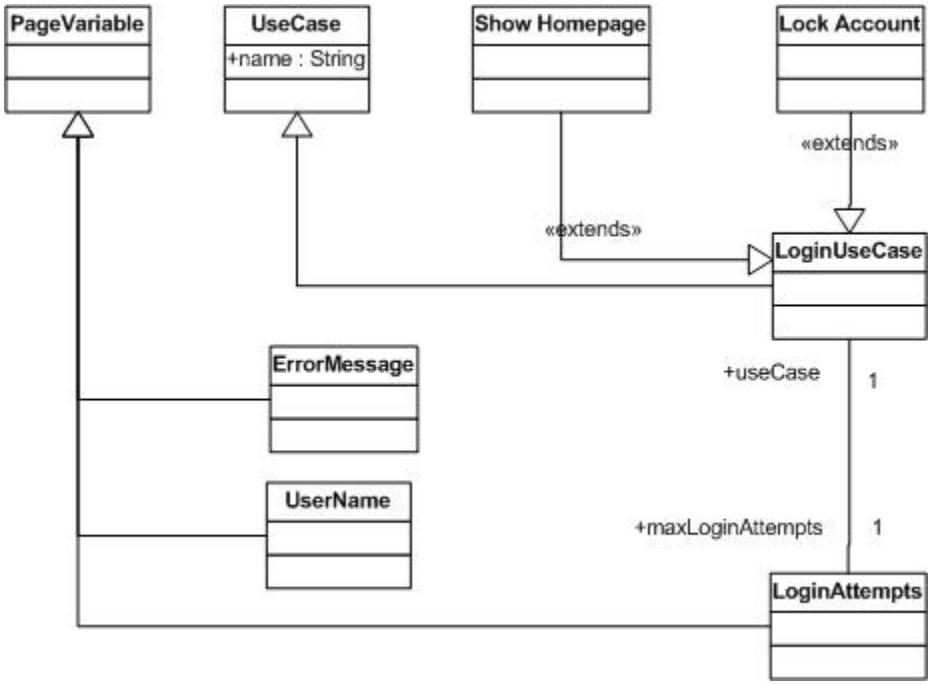

Figure 13 – Meta-Model support for the abstract login mechanism

In Figure 13, a state may enclose a presentation, which makes such state a *presentation state*. The attribute *login-required* applies to such states. An attempt to view a web page represented by those presentation states would result in an error message if the User is not logged in.

4.5. Page 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 following circumstances:

– To carry a default error message when the outgoing transition from a choice is labeled *false* . Such transitions are constrained as follows:

```
context falseTransition
        inv:
                self.guard.specification='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.

```
context presentationState
        inv:
                if self.loginRequired then self.userName->notEmpty();
```

– A page variable named *loginAttempts* is created when a login-required state is bound to a maximum number of login attempts.

Figure 14 presents the meta-model support for page variables.

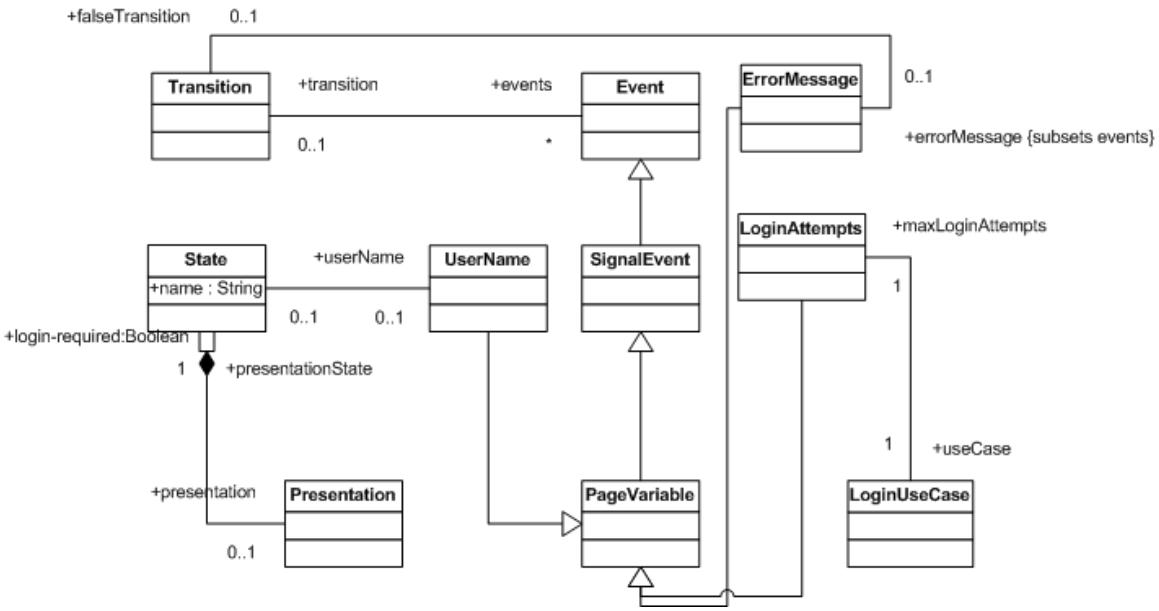

Figure 14 – Page Variable

Furthermore, 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.

4.6. Site Map

We suggest a default site map in concordance to the state machines. The application site map is created according to the following rules:

- For every final state associated with another use case, a link from the current use case to the first page of the annotated use case is created.
- Every presentation state is assessed to check the target of outgoing transitions. If the transition ends at another presentation state – regardless of other states in between – a link is generated from the source presentation state to the target presentation state.

Figure 15 provides the meta-model fragment for the default site map.

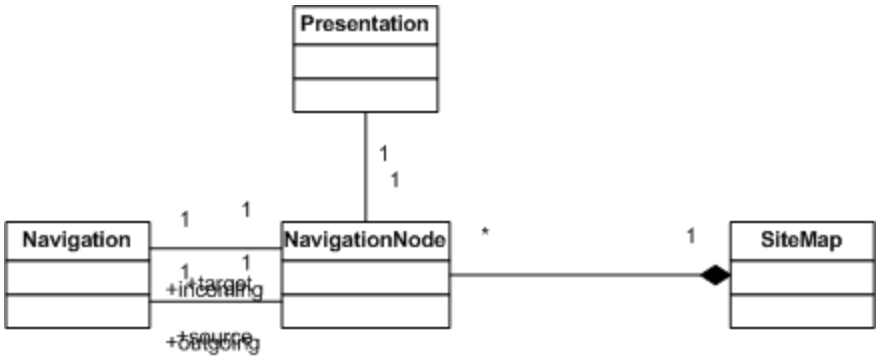

Figure 15 – Site Map Meta-Model

4.7. 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 different platforms. 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. Figure 16 shows this model.

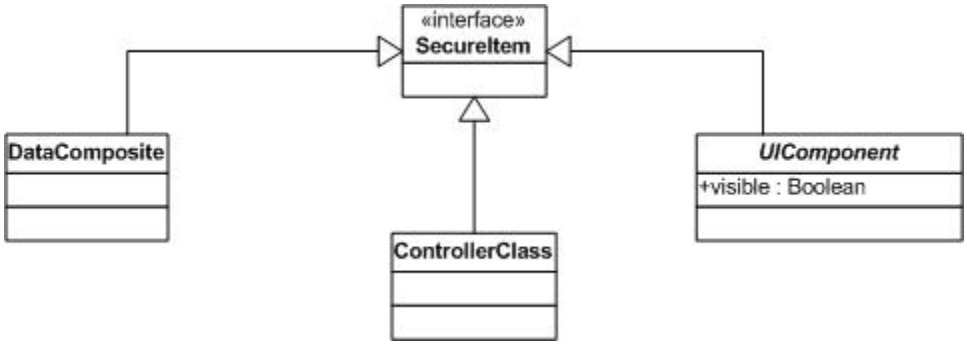

Figure 16 – Securing Data, Control and Presentation

4.10. A Multi-Level Meta-Model

The meta-model presented so far is capable of defining applications in at least three virtual layers. These are supplied by three core elements: *DataComposite*, *UIComposite* and *ControllerClass*, which represent model, view and controller in a traditional MVC pattern. This, by no means, lead to constraining applications to only three layers. As indicated in the meta-model, every data composite may be composed of other data composites. In the same way, a UI composite, being a UI component may include other UI composites. Also note that *UserInterface* is a type of *UIComposite*.

As an example, consider online tutorials provided in some web applications in order to teach certain processes. Such tutorials may be equipped with links given to specific tasks in each step. Figure 17 shows this situation. In this figure, *useCase1* is the scenario use case while *useCase2* is one of the steps. *Presentation1* is a UI related to *useCase1* and *presentation2* is the UI related to *useCase2*. In the same way, *dataComposite1* and

dataComposite2 are associated with *presentation1* and *presentation2* respectively. Figure 17 shows that a data composite composed of a second data composite – such as the contents of a tutorial as opposed to the contents of each section of it – may be recalled by two user interfaces at two different levels.

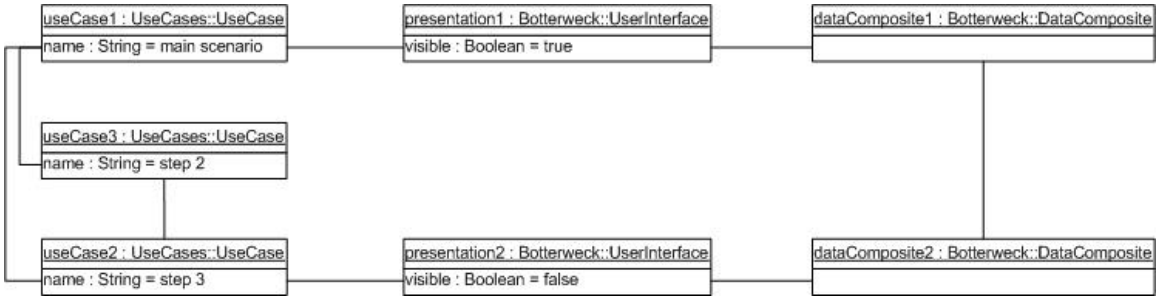

Figure 17 – Multi-level design of a web application

4.9. Specific Structures

Specific structures and procedures on web information systems are becoming popular and appear on most web sites nowadays. Some examples such as the login process and site maps were discussed. Other examples that are also supported are *Shopping Carts, Search, Feedback and Frequently Asked Questions (FAQ)*. Figures 18-21 show the model support for these features. Note that the key point is to provide a model that is abstract and yet web-specific; therefore different features may be supported at different levels of abstraction.

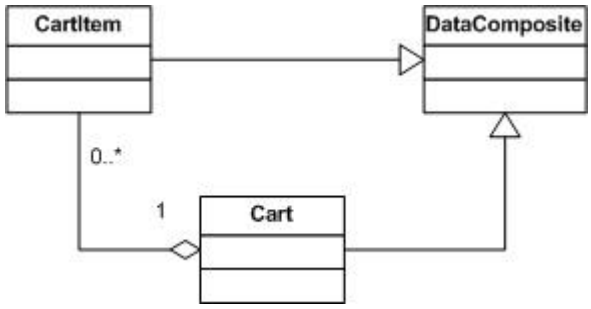

Figure 18 – Meta-Model of Online Carts

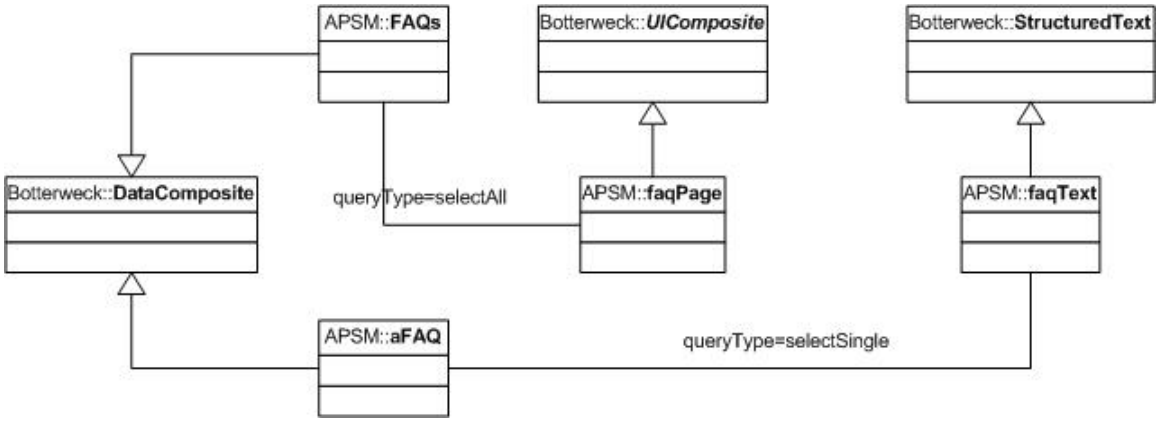

Figure 19 – Meta-Model of the FAQs

In Figure 19, both classes *faqPage* and *faqText* are of type *UIComposite*; that is they could both be associated with classes of type *DataComposite*. The type of query operation used for these associations is set to the default value *selectAll* for the class *FAQs* because the application shall load all items from the data item to the UI component. On the other hand, the type of query that associates the class *faqText* to the data composite *aFAQ* is *selectSingle* as each item will contain only one question.

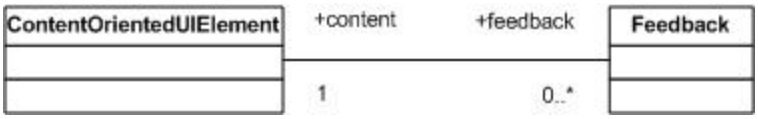

Figure 20 – Meta-Model of Feedbacks

Figure 21 – Meta-Model of the Search feature

4.10. Other Changes

Other changes that are noteworthy are as follows

- 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. This is performed using the model shown in Figure 22.

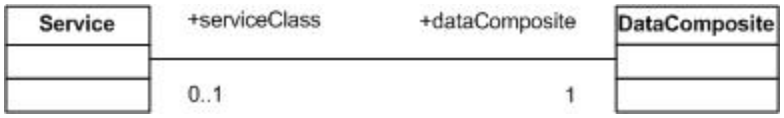

Figure 22 – Data Composite related to Service Class

- Every application requires one main use case to launch the process. We refer to this as the *startup use case*. The model is shown in Figure 23.

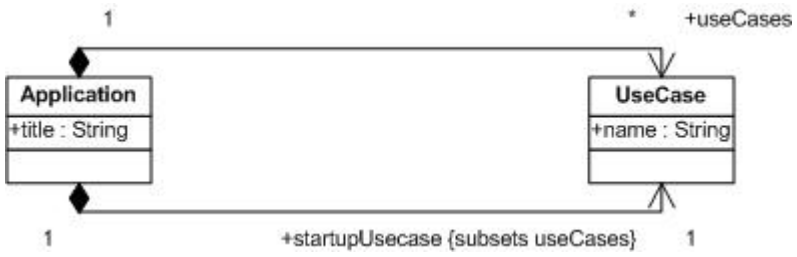

Figure 23 – Startup Use Case

• The behaviour of use cases is handled by use case controllers. The model needs a simple change to handle use case controllers. Figure 24 shows this facet from the meta-model.

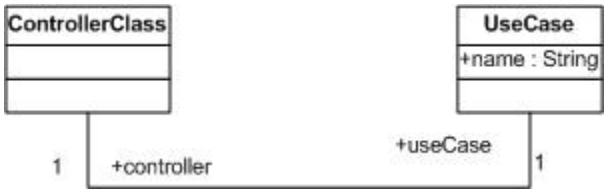

Figure 24 – Use Case Controllers

5. Proposing an Automated Model-Driven Method

Figure 25 depicts a general overview of a model-driven method for automated generation of web-based information systems using the meta-model presented in this report. The input (PIM) is provided as state machines representing a description of the system's use cases, as well as user interface (UI) prototypes. This input is then mapped to an abstract model of web-based applications. Meanwhile, we generate an abstract PSM consisting of classes required for the implementation of web applications such as controllers, entities and abstract information processing operations. In a second stage, the generated abstract web application is mapped to specific platform-specific models (SPSMs) chosen by the developer. In a third step that is not shown in Figure 25, the generated SPSMs are used to automatically generate executable applications using platform specific generators such as AndroMDA (2007) and WebRatio (2008).

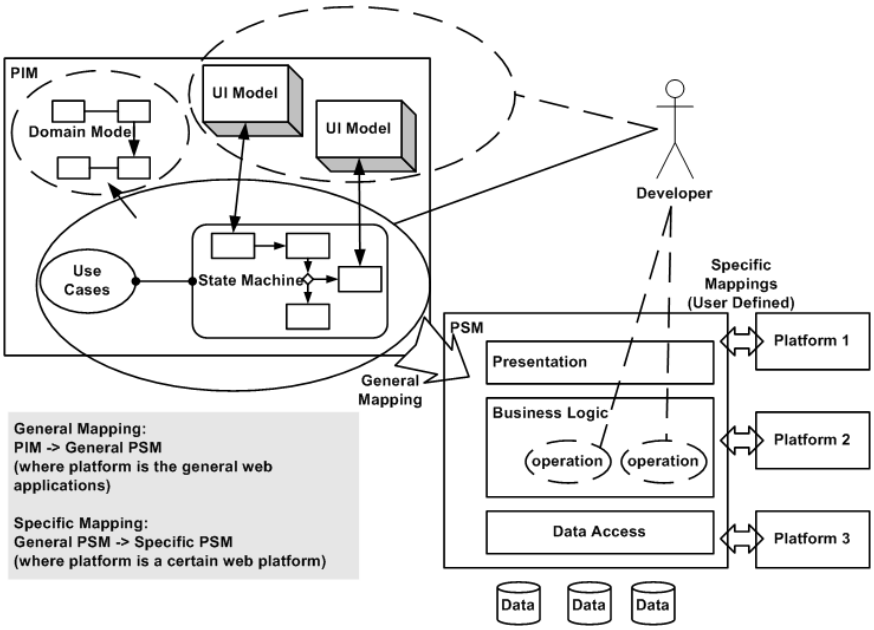

Figure 25 – Overview of the Proposed Method

5.1. Driving the Method: Deriving the Mappings

In this section, we review the rationale of designing the method of Figure 25 using our meta-model. We present the rationale to derive the method based on the model by dividing the model into two sections:

1. *pimS:* the set of elements required for building the PIM
2. *apsmS*: the set of elements required to generate the APSM

Since*pimS* is a subset of *apsmS*, the required transformations are *refining* transformations, in which the target meta-model has additional elements; a well-known example of such mappings are the conventional process of mapping analysis model to a design model (Mellor et al. 2004). This is the initial fact about the method driven by the model.

We continue deriving mapping rules by spanning the model. Figure 26 Presents a portion of the meta-model. Grey elements in Figure 26 represent *pimS*. In order to derive mapping rules, we need to identify the set *pimS'*, which is defined as:

$$pimS' = apsmS - pimS$$

In Figure 26, white elements belong to *pimS'; apsmS* is the union of both *pimS* and *pimS'*. Please note that the rules mentioned in this section are only introduced briefly as an indication of the fact that the method is driven by the model.

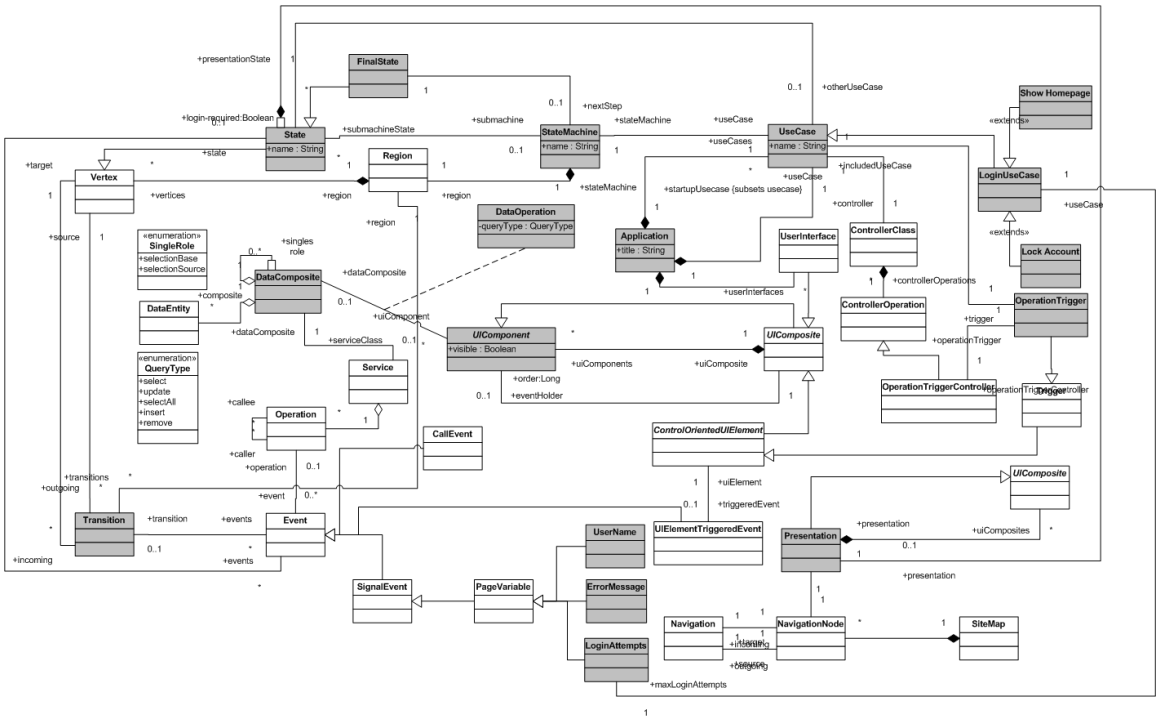

Figure 26 – PIM and APSM differentiated. Grey elements are only present in the PIM

In the rest of this section, we review the process of devising the mapping rules that transform elements from *pimS* to their counterparts from *apsmS*.

5.1.1. ControllerClass

The model shows a one-to-one relationship from controller classes to use cases. Thus, a mapping rule is required to map every use case to a controller class. In the same way, controller operations are derived from this fact that for specific situations, certain controller operations are suggested. For example, Figure 26 shows that associated with every operation trigger, there is a controller operation that handles the required logic and calls the desired data services. Therefore, a mapping rule is suggested to map every operation trigger to a corresponding controller operation.

5.1.2. Events

As Figure 26 shows, *pimS'* includes Events. Events include signal events and call events. Derivation of the transformations required to generate the events is as follows. Two possible paths may be found to discover events from the elements of *pimS*. First is to follow events through states and transitions but this will not lead to useful information mainly because it is not clear which transition to follow. An alternative path is found when we note that an operation trigger is in fact a *ControlOrientedUIElement* that is ultimately associated with *Event* as the supertype of all the events through the class *UIElementTriggeredEvent*. This leads to another mapping rule.

5.1.3. Services

A service class is associated with every data composite. This is simply generated based on the data composites that exist in association with UI components. Service operations

that are required to access and manipulate data are discovered by other mapping rules that are derived from the association class *DataOperation*, which relates every UI component to its corresponding Data Composite.

5.1.4. Data Entities

As the model suggests, data entities may aggregate through a data composite. Data composites also act as a proxy to data entities in order to carry the information required to access entities through different layers of the web application. Thus, a data composite could be composed of one or many data entities. In any case, this relationship leads to a mapping rule that yields to the generation of data entities based on the data composites associated to the UI components of a presentation state.

5.1.5. Page Variables

Figure 26 stresses that in the target model page variables are added to support special cases. There are direct connections between elements such as *LoginUseCase* or *State* and *PageVariable*. This is simply supported by concrete mapping rules that lead to the generation of required page variables in case the constraints become valid in the source model.

5.1.6. Site Map

Figure 26 draws the fact that a site map is generated in a close relationship with the class *Presentetaion*. Mapping rule is simply derived from the fact that every presentation is associated with a navigation node.

6. An Example

In this section, we illustrate the usage of the meta-model using an example. The example is taken from an Election Management System (EMS) introduced by Lethbridge and Laganière (2001) that has been used as one of our major validation projects. The intent of this system is to manage the information of elections and polls. The system provides facilities for data-related activities and management of the voting process. It also provides some information to journalists.

In this section, we review a core use case of the EMS that is the use case *Vote*. The description of this use case is as follows:

***Use Case**: Vote*
***Precondition**: Poll is Open*
***Steps**:*

1- Voter enters personal information
2- System shows empty ballot
3- Voter assigns candidates to seats
4- Voter confirms ballot
5- System prints ballot
6- System prepares voting screen for the next voter

***Alternatives**:*

1-a- Voter is not able/willing to enter his/her information
1-a-1- Election officer enters Voter's information
1-a-2- goto 2
3-a- Voter is not able/willing to vote electronically
3-a-1- System prints ballot
3-a-2- Voter fills in the ballot
3-a-3- Voter drops ballot in provided slot
3-a-4- goto 6

***Extension Point**: 2-a- Voter is not registered*

***Exnteding Use Case**: Register Voter at Station*

***Postconditions**:*

- Ballot is printed
- Ballot information is added to the poll database
- Default voting screen is shown

Figure 27 depicts a state machine that realizes the behavior of the use case *Vote*. The alternative to step 1 is not explicitly shown in Figure 27 because it does not change the behavior of the system but only the actor of a specific operation.

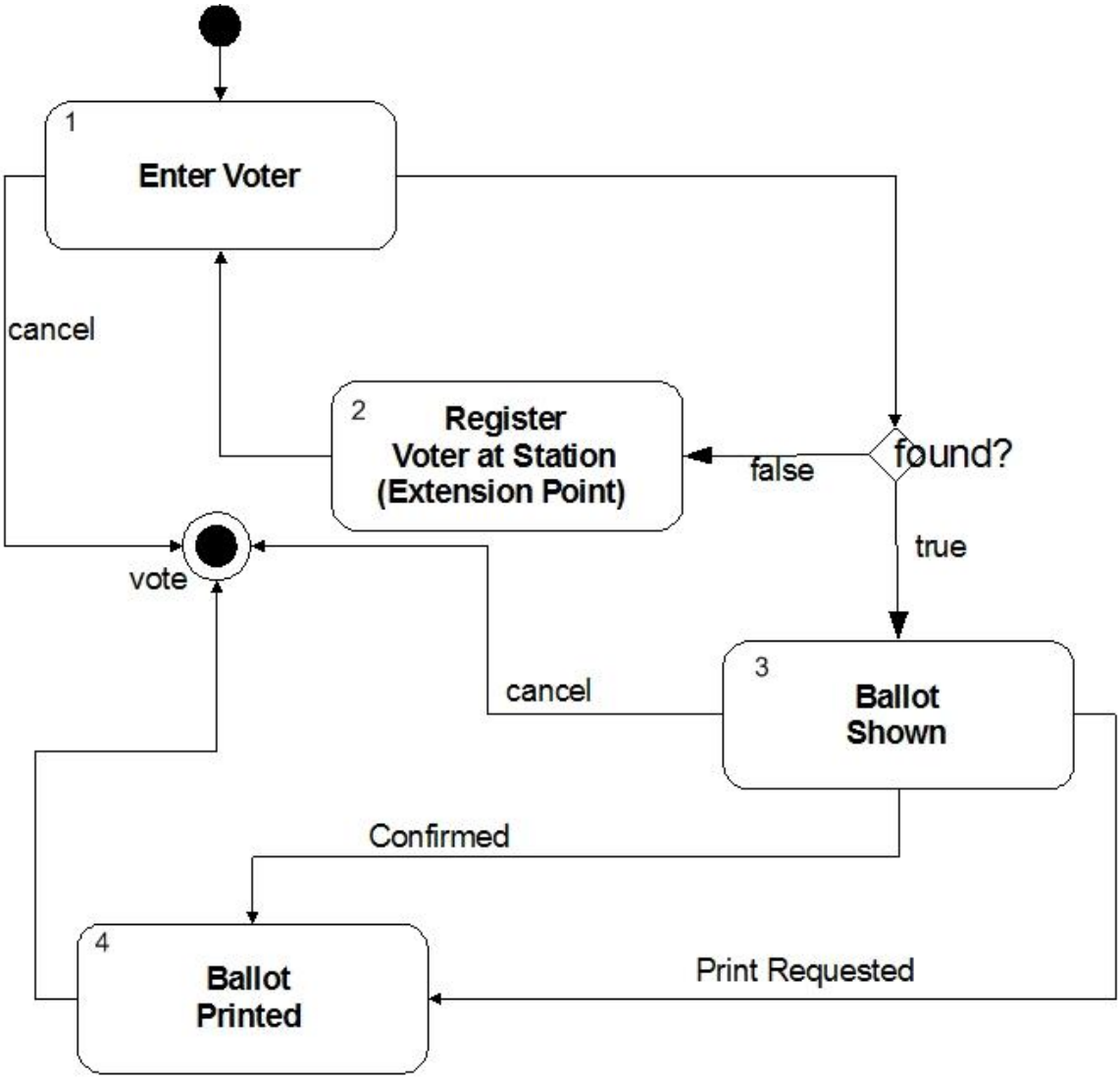

Figure 27 – The State Machine *Vote*

Figure 28 provides the presentation model of state 1. A voter or an election officer may enter voter's information in order to login. Alternatively, the button *Register* may be clicked for registering the Voter at the station if required. As Figure 28 suggests, the button *Show Ballot* activates a query over the existing records of the entity Voter.

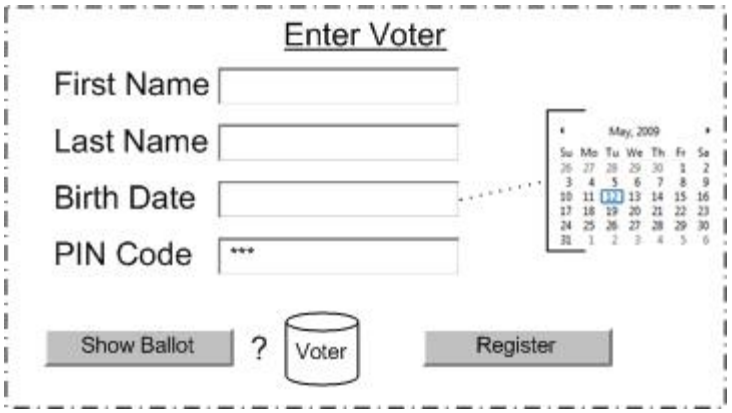

Figure 28 – Presentation of State *Enter Vote*

Figure 29 presents the presentation of the state 3. The model encompasses the alternative to step 3 to print the ballot for manual voting.

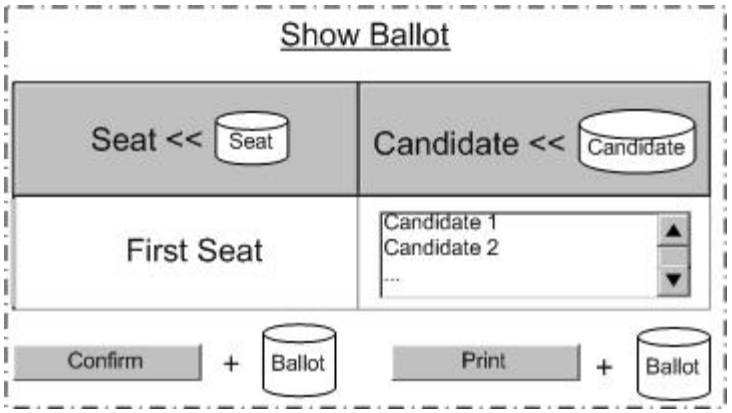

Figure 29 – Presentation of State *Show Ballot*

6.1. The Model of The Input

In this subsection, we review the object model that supports the input model including the state machine, use cases and the presentations. Figure 30 presents the model using a UML object diagram. This model provides an image of the models provided in Figures 27-29. Only links between use cases are unidirectionally annotated in order to emphasize the relationships between use cases. Transitions are not presented in this model in favor of brevity.

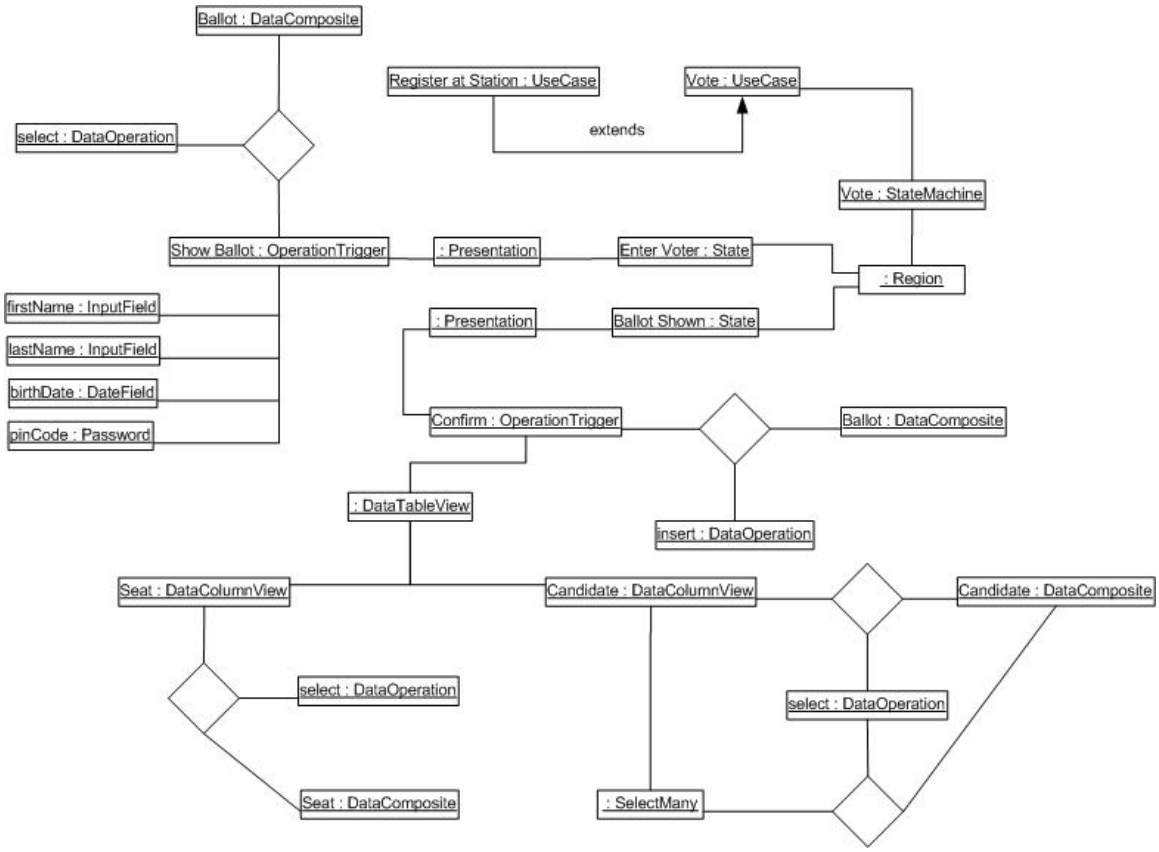

Figure 30 – The Object Model of the Input

6.2. The Model of the Output

In this subsection, we review the application of the mapping rules to the model depicted in Figure 30. Since the output model encompasses too many details, an object model of the whole output does not reveal useful information. Instead, we review a partial output model resulted from the application of a few mappings. This is shown in Figure 31.

Three controller classes are created for three use cases in the model as described by the mapping rules. Another mapping rule requires the generation of controller operations triggered by the events related to the events from the screen. The operation *showBallot* is generated in the controller *showBallotController* as one of the mapping rules suggests to support the operation trigger *Show Ballot*. The same contribution creates the operation *confirm*.

Related to each operation trigger, a signal event is created on the outgoing transition leaving the corresponding presentation state. As an example, Figure 31 shows the signal event created on the outgoing transition of the state *Show Ballot*. The model presents an automatically added state – that has no name – which is inserted after the presentation state *Show Ballot* to satisfy the constraints of the model. A call event is created on this state calling the controller operation *showBallot()*. This operation has four parameters matching the input fields of the form *show ballot*. The *showBallot* operation calls two service operations for selecting all relevant seats and candidates from the data service classes generated upon the existence of data composites Seat and Candidate.

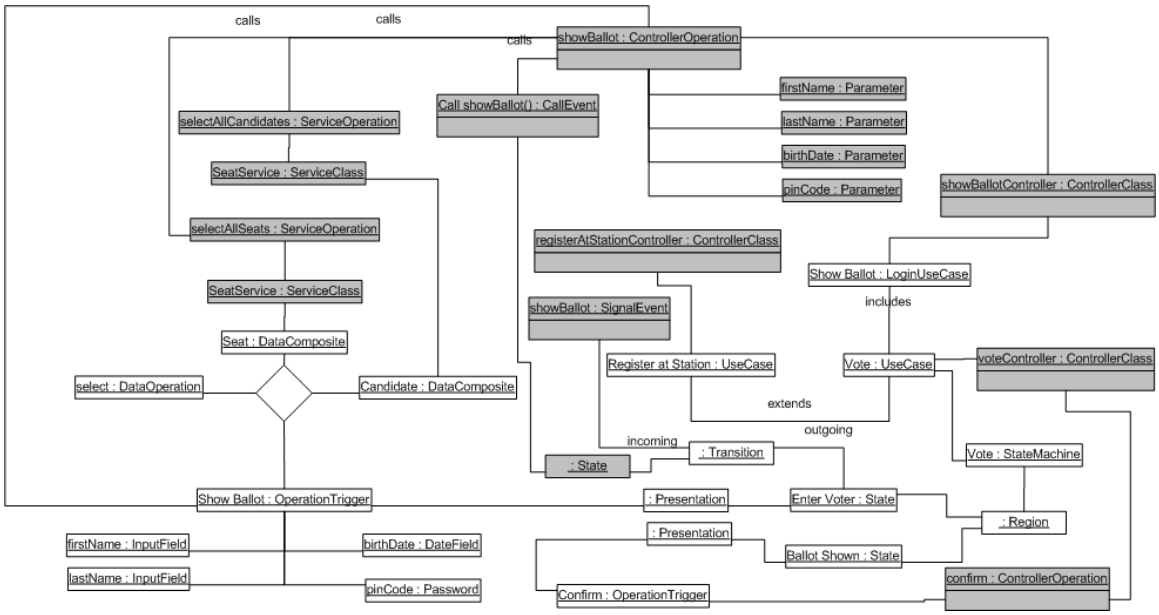

Figure 31 – An excerpt of the output Object Model

7. Conclusion

In this document, we presented a meta-model for web information systems. This meta-model was built upon a survey of the existing models, which led to the selection of one of those models. The selected model was then extended because of the need to cover more concrete features of web information systems. The model provides elements to model different aspects of a web application at an abstract level. Models instantiated from this meta-model will bear many of the common elements of web applications without being dependent to any specific web platform.

The model presented in this report has been tested for a number of example systems. We have implemented 23 use cases for the EMS. Three other major case studies were used for validation purposes: A Team Management System (TMS) by Somé (2008) has been a course project with 15 use case for assisting instructors to manage student teams. An Account Management System (AMS) that is a foreseen project in a waste/water management company to manage customer accounts, which replaces an existing non-web system with 42 use cases. Finally, a member card sub-system similar to the one used in many businesses with 15 use cases Different case studies are chosen carefully in order to cover different aspects of modeling systems. For example, the member card sub-system contains certain occasions of processing more than one instance of the same entity in a single presentation unit.

In order to perform validation/evaluation, an eclipse-based environment was created and tested upon several case studies. The meta-model and instance models are created using the Eclipse graphical modeling framework (GMF). Since our main output is an abstract-platform specific application, and not the executable application itself, the output was transformed to two different platforms in order to evaluate the suitability of the generated APSM. The platforms are AndroMDA (2007) and WebRatio (2008). Also, instances of the meta-model regarding each case study were implemented using Eclipse Modeling Framework (EMF, 2009). Sets of transformations to map the meta-model to the above mentioned platforms were also successfully implemented using the eclipse-based tool MediniQVT (Medini 2008).

Further work would be required to map the abstract model to more specific platforms. This will ensure the model is positioned at an appropriate level of abstraction. The development of utilities to map the models built upon this meta-model to other specific platforms will also be another part of the future work.

References

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

Baresi, L. Colazzo, S. Mainetti, L. and S. Morasca. W2000: A Modeling Notation for Complex Web Applications. In E. Mendes and N. Mosley (eds.) Web Engineering: Theory and Practice of Metrics and Measurement for Web Development. Springer, ISBN: 3-540-28196-7, 2006.

Beydeda, S. Book, M. Gruhn, V. (eds.).Model-driven software development, Berlin ; New York : Springer, 2005.

Blankenhorn, K. A UML Profile for GUI Layout, Master's Thesis, University of Applied Sciences Furtwangen, Department of Digital Media May 23, 2004

Bogdan, C. Falb, J. Kaindl, H. Kavaldjian, S. Popp, R. Horacek, H. Arnautovic, E. and Szep, A.: Generating an Abstract User Interface from a Discourse Model Inspired by Human Communication. HICSS 2008: 36

Botterweck, G. A Model-Driven Approach to the Engineering of Multiple User Interfaces, In: Models in Software Engineering, , Springer Berlin / Heidelberg, Volume 4364/2007, pp. 106-115

Botterweck, G. 2007. 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 http://www.amazon.de/Multi-Front-End-Engineering-modellgetriebener-Entwicklung-Anwendungen-Fronts-Ends/

Brambilla,M. Fraternali, P. Tisi, M. "A Metamodel Transformation Framework for the Migration of WebML Models to MDA". 4th Int. Workshop on Model-Driven Web Engineering (MDWE 2008). N. Koch, G.-J. Houben, A. Vallecillo (Eds.). CEUR Proceedings, volume 389, pages 91-105.

Costa, D. Nóbrega L. and Nunes, N. J. An MDA Approach for Generating Web Interfaces with UML ConcurTaskTrees and Canonical Abstract Prototypes, 137-152. In Book: Task Models and Diagrams for Users Interface Design

Ceri, S. Fraternali, P. and Bongio, A.: Web Modeling Language (WebML): a modeling language for designing Web sites. Computer Networks 33 (1-6): 137-157 (2000)

Cockburn, A. Writing Effective Use Cases. Boston : Addison-Wesley, c2001.

da Silva P. P. and Paton, N. W.: Improving UML Support for User Interface Design: A Metric Assessment of UMLi. ICSE Workshop on SE-HCI 2003: 76-83

de Souza, R. A. C. de Barros, R. S. M. A Model-Driven Method for the Development of Web Applications User Interaction Layer. TASE 2008: 91-98

Diamodl, http://www.idi.ntnu.no/~hal/research/diamodl, July 19, 2008

Epner, M.: Poor project management number-one problem of outsourced eprojects.Research Briefs, Cutter Consortium (2000). Available from: http://www.cutter.com/research/2000/crb001107.html

EMF, http://www.eclipse.org/modeling/emf/, September 2009

Freudenstein , P. Nussbaumer , M. Allerding , F. Gaedke, M. A domain-specific language for the model-driven construction of advanced web-based dialogs. Proceeding of the 17th international conference on World Wide Web, Pages 1069-1070, 2008

Google Web Toolit, Google Code, http://code.google.com/webtoolkit/, June 2010

He, C. Tu, W. and He, K. Role Based Platform Independent Web Application Modeling Proceedings of the Sixth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'05). Pp: 411-415.

Hruby, P. Model-Driven Design using Business Patterns; with contributions by Jesper Kiehn and Christian Vibe Scheller. Berlin ; New York : Springer-Verlag, c2006.

Jinkui, H. Jiancheng, W. Yongtang, Y. A Semantics-Reconstruction Based Model-Driven Development Approach for Web Information Systems Chinese Control Conference, 2007. pp: 344-348

Kavaldjian, S. A Model-Driven Approach to Generating User Interfaces, Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, Doctoral symposiumPages: 603 – 606, 2007

Kroiß, C. and Koch, N. UWE Metamodel and Profile User Guide and Reference, Technical Report, February 2008. Available from (www.pst.ifi.lmu.de/projekte/uwe)

Lethbridge T. C., Laganière R., Object-Oriented Software Engineering: Practical Software Development using UML and Java, London : McGraw-Hill, 2001

Li, J. Chen, J. Chen, P.: Modeling Web Application Architecture with UML , Proceedings of the 36th International Conference on Technology of Object-Oriented Languages and Systems, 2000. TOOLS - Asia 2000. 265-274

mediniQVT – Trac, http://projects.ikv.de/qvt, 3 May 2008

Mellor, S. J. Scott, K. UHL, A. Weise, D. MDA Distilled : Principles of Model-Driven Architecture. Boston : Addison-Wesley, c2004.

Moreno, N. Fraternalli, P. Vallecillo, A. A UML 2.0 profile for WebML modeling Workshop proceedings of the sixth international conference on Web engineering, Second international workshop on model driven web engineering (MDWE'06) 2006

Moreno, N. Fraternali, P. Vallecillo, A. WebML modeling in UML. IET Software Journal (2007)

Muller, P. A. Studer, P. and Bézivin, J. Platform Independent Web Application Modeling. In: "UML" 2003 - The Unified Modeling Language, Springer Berlin / Heidelberg, Volume 2863/2003, Pages 220-233

Nikolaidou, M. and Anagnostopoulos, D. A Systematic Approach for Configuring Web-Based Information Systems. In the Journal of Distributed and Parallel Databases, Springer Netherlands, Volume 17, Number 3 / May, 2005 .Pages 267-290

Nunes, D. A. and Schwabe, D.: Rapid prototyping of web applications combining domain specific languages and model driven design. ICWE 2006: 153-160

OMG, MDA Guide Version 1.0.1, 12th June 2003

OMG, Meta Object Facility (MOF) Specficiation Version 1.4, April 2002

OMG, Unified Modeling Language: Superstructure, February 2007

IBM – Rational Unified Process (RUP), http://www-01.ibm.com/software/awdtools/rup/, September 2009

Schauerhuber, A. Wimmer, M. Kapsammer, E. Schwinger, W. Retschitzegger W. (2007). Bridging WebML to model-driven engineering: from document type definitions to meta object facility, IET SOFTWARE, 1-3, p. 81 - 97

Schattkowsky, T. and Lohmann, M. Towards employing UML Model Mappingsfor Platform Independent User Interface Design. Springer Berlin / Heidelberg. Volume 3844/2006, Satellite Events at the MoDELS 2005 Conference Pages 201-209

Schmidt, D. Stal, M. Rohnert, H. and Buschmann, F. Pattern-Oriented Software Architecture. Chichester [England] ; New York : Wiley, <2000-c2007>

Stéphane Sotèg Somé's homepage, http://www.site.uottawa.ca/~ssome/, Fall 2008.

Sousa, K. S. Filho, H. M. Vanderdonckt, J.: Towards Method Engineering of Model-Driven User Interface Development. TAMODIA 2007: 112-125

Stahl, T. and Volter, M, Bettin, J. Haase, A. and Helsen, S.: Modeldriven Software Development : Technology, Engineering, Management /translated by Bettina von Stockfleth. John Wiley, Chichester, England ; Hoboken, NJ (2006)

Sukaviriya, N. Sinha, V. Ramachandra, T. Mani, S.: Model-Driven Approach for Managing Human Interface Design Life Cycle. MoDELS 2007: 226-240

Tongrungrojana, R. Lowe, D.: WIED: A Web Modelling Language for Modelling Architectural-Level Information Flows. J. Digit. Inf. 5(2): (2004)

Tongrungrojana, R. Lowe, D.: WebML+: a Web modeling language for forming a bridge between business modeling and information modeling. SEKE 2003: 17-24

? David Lowe, Rachatrin Tongrungrojana: WebML+ in a Nutshell: Modeling Architectural-level Information Flows. WWW (Posters) 2003

UWE,-UML based Web Engineering, http://uwe.pst.ifi.lmu.de, 2-3-2008

Vanderdonckt, J. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In : Advanced Information Systems Engineering, Volume 3520/2005, Springer Berlin / Heidelberg, Pages 16-31.

Wan, J. Bieber, M.: GHMI: A General Hypertext Data Model Supporting Integration of Hypertext and Information Systems. HICSS (2) 1996: 47-

Wan, J. Bieber, M. P. Wang, J. T. L. Ng, P. A.: LHM: a logic-based hypertext data model for integrating hypertext and information systems. HICSS (3) 1995: 350-

WebML , www.webml.org, 2-1-2008

WebRatio, www.webratio.com, 2-2-2008

Whitehead Jr. E. J., Ge, G. Pan, K.: Automatic generation of hypertext system repositories: a model driven approach. Hypertext 2004: 205-214