

## MODEL-DRIVEN WEB DEVELOPMENT FOR MULTIPLE PLATFORMS

ALI FATOLAHİ

*University of Ottawa*

*School of Information Technology and Engineering*  
*afato092@site.uottawa.ca*

STÉPHANE S. SOMÉ

*University of Ottawa*

*School of Information Technology and Engineering*  
*ssome@site.uottawa.ca*

TIMOTHY C. LETHBRIDGE

*University of Ottawa*

*School of Information Technology and Engineering*  
*tcl@site.uottawa.ca*

Received April 28, 2009

Revised May 17, 2011

Model-driven development of web applications relies on the definition of the mappings that transform high-level models to models of specific web platforms. Thus, the transformations are often platform-specific and may not be used for more than one platform. The current web, however, is a heterogeneous network of different technologies and it often happens that one specific application needs to run on several platforms. Also, many patterns of web applications could be re-used in several projects that are performed using different technological configurations. In this paper, we describe our approach for targeting multiple platforms by defining an intermediate abstract web platform. This is a technology-independent model that carries common properties of web applications. Thus, transformations will become two-step transformations; the first step targets the abstract web platform and hence, is re-usable. The second step maps the abstract web model to specific web platforms; this is shorter than conventional platform-specific transformations.

*Key words:* MDD, Web, Abstract PSM, Transformation, UI Model, QVT Relations  
*Communicated by:* D. Lowe & N. Koch

### 1 Introduction

Web applications are increasingly prevalent in various application areas including commerce, health, communication and education. Applications are being developed and deployed at a very fast pace to cope with changing business requirements and opportunities. Moreover, the technologies upon which web applications are based are also subject to rapid evolution. The quick rate of change in web development technologies highlights the need for methods that generate

reusable models, which are independent of target platforms. In this paper, a model-driven approach aimed at automating a part of the development process of web-based information systems is introduced. A goal of the approach is to increase the level of re-usability by helping avoid repeating similar tasks.

We are specifically concerned with the technology-related difficulties pointed out by Shklar and Rosen [1], Meliá and Gómez [2] and Ceri et al. [3]. These mainly refer to the variety of web platforms and challenges related to configuring applications for these platforms. The diversity of technologies and the quick rate of incoming new technologies also affect the cost and time of web development because of the need to learn new technologies and development processes. Technology-related problems also include the difficulty of separation of concern between platform-specific features and those common to all web platforms [4-7].

In this paper, we propose a model-driven approach for the generation of a platform-specific web model from a high-level platform-independent model. The approach is based on the Model Driven Architecture (MDA) [8]. The input, termed a Platform-Independent Model (PIM), consists of use case descriptions of the application as abstract state machines and user interface prototypes. The result of the approach, termed a Platform-Specific Model (PSM), is a design model detailing features of web applications for a specific platform that is obtained in two steps:

- An abstract PSM (APSM) is generated in the first step, which is defined upon an abstract model of 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.
- The APSM is then mapped to Specific PSMs (SPSMs) in the second step; SPSMs describe concrete web-based platforms such as J2EE or .NET [9].

This is very similar to the notion of abstract machine used by the Java technology.

Different benefits can be foreseen from our approach. By relieving developers from low-level platform specific related design, the approach has the potential to shift the development task to issues related to business needs. Another benefit is the shortened 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 [10]. The approach is specifically suitable for information-intensive web-based systems. These applications typically involve large data stores accessed through a web interface. A distinctive aspect of our approach is its use of a specification of the data mapping as part of the PIM. More importantly the common features required to process data and communicate data objects between different layers and components are targeted.

The rest of this document is organized as follows. In Section 2, we provide an overview of our approach, models and techniques. Section 3 describes the meta-model used. In Section 4, we specify our approach in terms of the mapping rules. Section 5 contains a case study. Section 6 elaborates the implementation environment of the approach and the models. Section 7 reviews the results of using the approach in real projects. Section 8 contains a discussion of the related work. Finally, Section 9 concludes the paper with a focus on future work.

## 2 Overview of Automated Web-Application Generation

Figure 1 depicts a general overview of our proposed approach. The input (PIM) is provided as state machines representing a description of the system’s use cases, as well as user interface (UI) prototypes. The dashed line connecting the developer to this step asserts that this step is semi-automated. Based on this input, we generate an abstract PSM (APSM) consisting of classes required for the implementation of web applications such as the controllers, entities and abstract information operations. In the second stage, the generated abstract web application is mapped to concrete platform-specific models (SPSMs) chosen by the developer. A resultant SPSM is eventually used to automatically generate executable applications for specific web platforms.

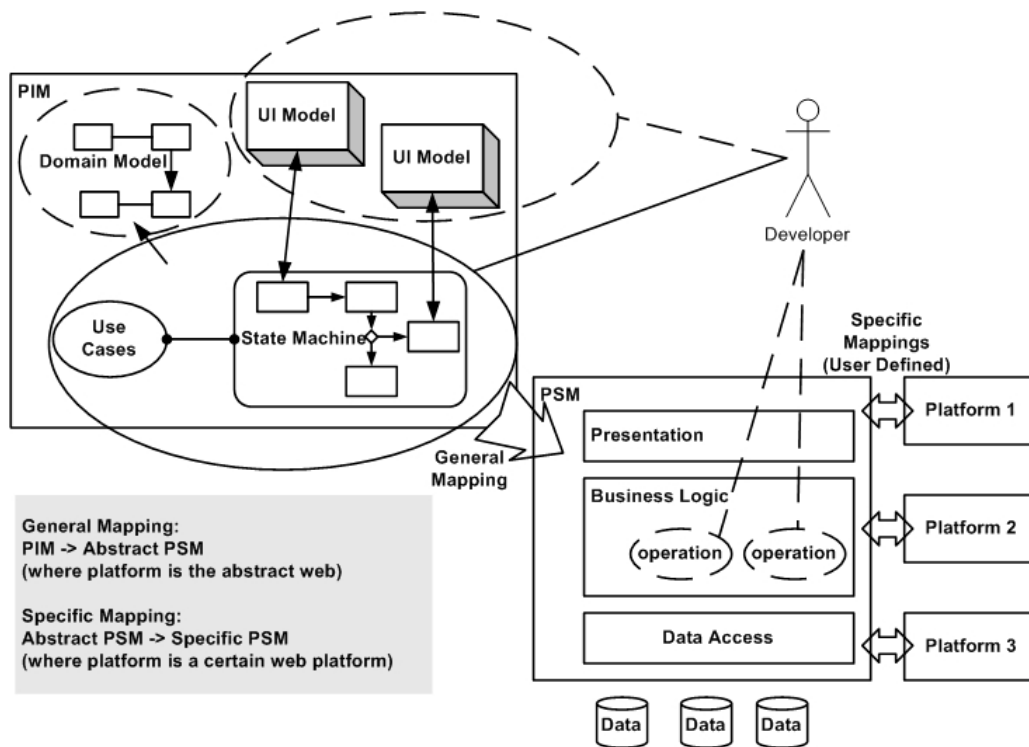


Figure 1 Overview of the Proposed Approach

We remind the reader that the term platform in our approach refers to the notion of abstract web. Our PIM is, therefore, not platform dependent as it does not contain details specific to common web platforms. The APSM is dependent on the web as an abstract platform without any relation to a specific technology. The addition of more details related to specific platforms leads to models dependent to specific web platforms (SPSMs).

The following are components of our PIMs:

- UML use case models with UML state machines describing the behavior of use cases. PIM level state machines do not need to be elaborated. Particularly, our transformations only rely on transition names for generating actions, events and guards at the APSM level.
- User interface models, which specify the desired user interface and related information elements.

The generated abstract application (the APSM) is a general web application with behaviour definitions for the processing flow and additional structures. The APSM generated model also includes common features found in web information applications including: Login/Logout Features, Online Carts such as shopping carts or favourite lists found within online libraries, Search Features, Support for Comment/Feedback, and Support for Frequently Asked Questions. These features are proposed as selectable options for inclusion in modelled applications. More specifically, elements of the APSM are:

- The State machines from the PIM enriched with state and transition events, operation calls and parameters.
- The UI model and its components.
- Data elements and components used for information storage and retrieval, and to support the UI components.
- Definition of controller, service and entity classes.
- Definition of operations used for information processing.

The contents of SPSMs are different from platform to platform; an example will be discussed in Section 4.2. The extent of generated code for the concrete application depends on the specific platform used. Operations for CRUD-based access are normally fully generated. These are operations that involve interaction with a database system for: Creating a new record of data, Reading a set of data instances according to a defined criteria, Updating specific elements of data according to certain criteria and a change pattern, or Deleting an existing data object. Nevertheless, code generation is out of the scope of our work and is only performed for the platforms that are used as example targets of the approach.

### **3 Web-Application Modelling**

Transformations performed in Figure 1 require the definition of source and target models based on a meta-model. Figures 2-5 present the most important parts of our meta-model. Our model is partly adapted from [11]. Notice that the same meta-model is used to express both the PIM (source) and PSM (target) models. This is due to the fact that our PSM is basically a detailed version of the PIM.

Figure 2 shows the general structure of an application in our approach. The application has several use cases, a few of these 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



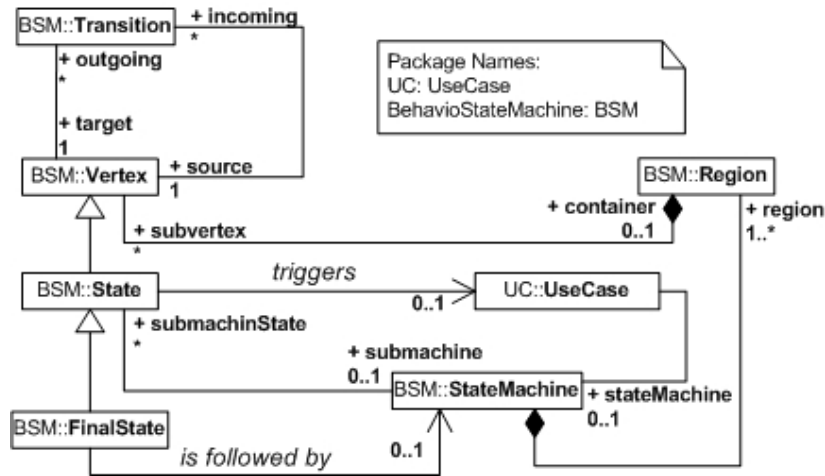


Figure 3 The Meta-Model of Web Information Systems, Use Cases and State Machines

Figure 4 shows how our meta-model supports the presentation and data in one model. States may be associated with presentations; this results in a presentation state. A presentation is a special UI Composite, which means it could contain other UI Components. UI Components may be associated to each other through a *FieldOperation*. 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 the basis of selecting data from another one. For example, when selecting a country affects the list of available provinces such an association would be created.

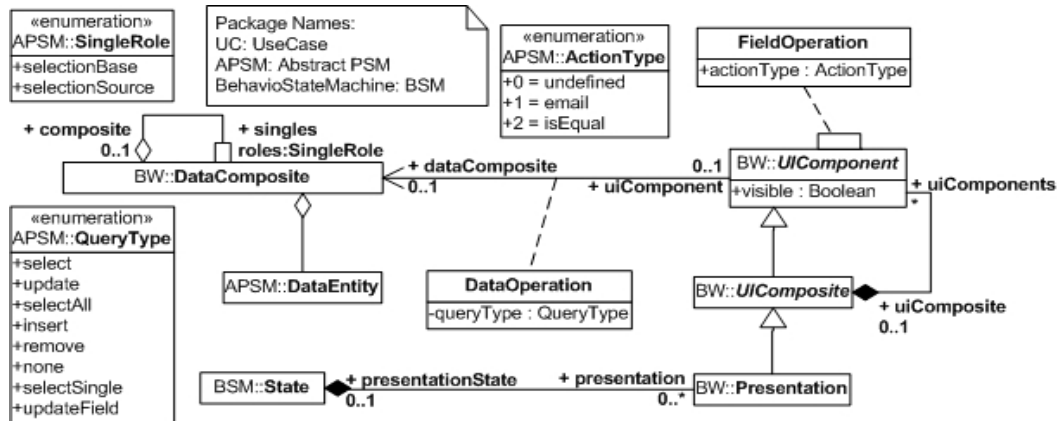


Figure 4 The Meta-Model of Web Information Systems, Presentation vs. Data

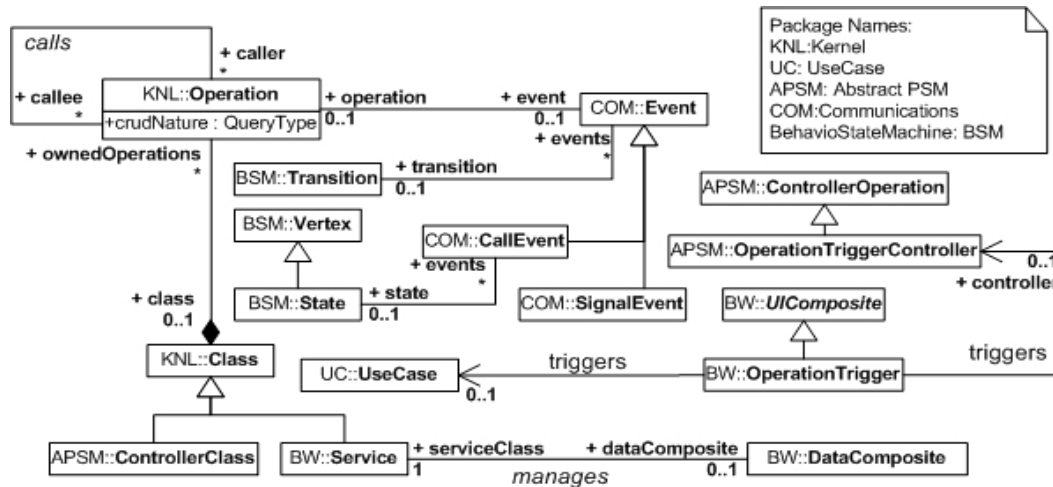



Figure 5 The Meta-Model of Web Information Systems, Events and Operations

Finally, Figure 5 presents the part of the model, where events and operations are handled. Every data composite is supplied with a service class, which manages the flow of information to and from the data composite. An attribute *crudNature* is added to the class *Operation* in order to determine the type of service operation and is used for code generation. Also, in order to model the call structure from controller operations to service ones, an association is added from the class *Operation* to itself. A special UI composite, *OperationTrigger*, is used to represent information submitted to the server side, for example, as part of web form. Operation trigger may cause either an included use case or a controller operation to be triggered.

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. In any case, the developer is never required to assign elements to specific layers. In other words, modeling the application is a process of defining as many elements as required and relating them to each other without any specific concern of the number of layers.

Several modeling elements in Figures 2-5, extend those the original Botterweck meta-model; this applies to all semantic and structural aspects. The Figures do not repeat many elements from the UML meta-model simply because doing so would not be meaningful in the context of our approach. For example, initial states as one kind of pseudo-states are not shown. However, they are still part of the meta-model and are used in generated models for the sake of consistency with other methods, tools and models.

Aside from the abstract model, the following graphical notation is used to specify the relationship of data elements to UI components in presentations:

- , A data source that could be equivalent to a single database table or a composite object.
- ?, Used to associate a data source to a UI component denoting that the component will cause a filtered selection of instances of the data object. The filter is determined based on the type of UI component.
- +, Used to associate a data source to an operation trigger inferring that the trigger will fire an insertion operation.
- <<, Used to associate a data source to a UI component denoting that the component will cause the selection of all instances of the data object.
- >>, Used to associate a data source to an operation trigger inferring that the trigger will fire an update operation.
- -, Used to associate a data source to an operation trigger inferring that the trigger will fire a deletion operation.
- \*, Attached to any UI Component; asserts that the component is the one that fires the default event. Usually, the default event is kept on the operation trigger. This is a mechanism to switch the event to other elements such as a drop-down select box that fires a selection event.

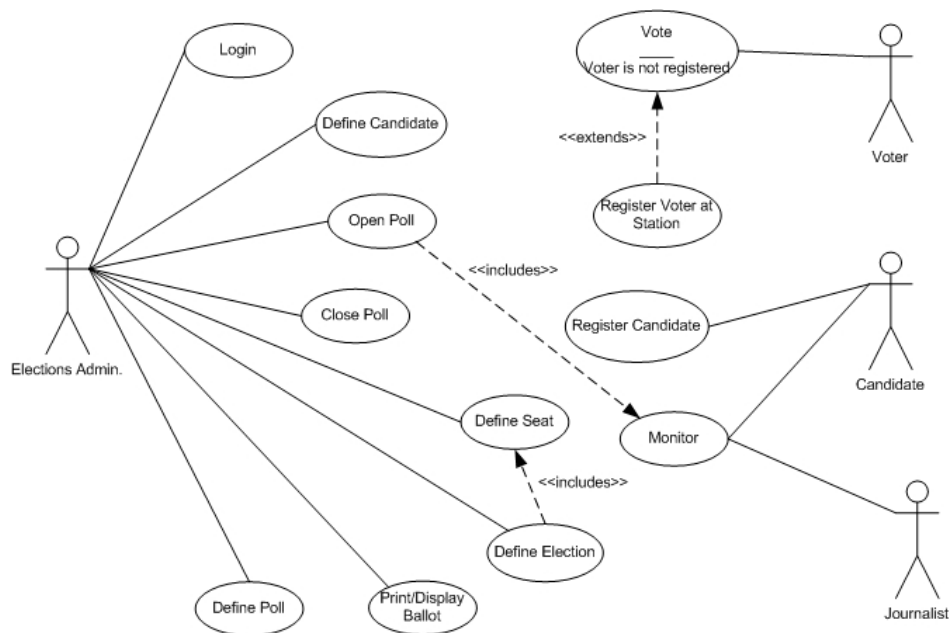


Figure 6 Core Use Cases of EMS



An Election Management System (EMS) [12] is used to illustrate the approach. The EMS is intended to assist election bodies to set up and conduct elections at different levels. The EMS is an interesting case study in the sense that it involves several associations among data elements and has use cases of significant complexity. Figure 6 shows a use case diagram for the EMS, which includes core use cases. Other use cases such as those for handling CRUD operations regarding different data elements are not included in this diagram for conciseness.

#### 4 Mappings

We present our PIM to PSM mappings in this section. As previously mentioned, these mappings are performed in two steps. In the first step, we generate an Abstract PSM (APSM), which is thereafter mapped to Specific PSMs (SPSMs). The mapping rules from PIM to APSM are discussed in Section 4.1 and mapping rules from the APSM to a particular SPSM in Section 4.2. These mapping rules have been formally defined as QVT relations. A list of a set of QVT relations that lead to a level of meaningful APSM elements is provided in Appendix A.

Most of the state machines drawn in Sections 4 and 5 are PIM state machines unless precisely specified. Being UML Namespaces - and thus NamedElements -, transitions are named in our PIM state machines and only the provided names are relevant to APSM generation. Transition names will be eventually mapped to the names of the corresponding guards, events and triggers that are automatically added at the APSM level.

It is critical to understand that PIM state machines - being only inputs - do not need to be formally complete because they will not be used as the basis of detailed design or implementation. On the other hand, APSM state machines are fully formalized and include all the details a developer - or an automated approach - requires finding detailed design and implementation.

We consider the EMS use case *Open Poll* as running example. Figure 7 shows a state machine that models the behaviour of use case *Open Poll* along with attached UI models. The state machine in Figure 7, involves two presentation states; that is, the use case communicates with the User in two steps. The following is the list of states:

- **Open Poll view is shown:** In this presentation state, the User selects an election from a drop-down select box. The star on the Election select box denotes the default component that causes the outgoing transition. The “<<” mark along with the cylinder named, Election asserts that the content of the component comes from the data source, Election.
- **Populate Polls List:** The drop down select box for Polls is populated in this state. This will be done using the Election ID that binds the selection of polls to those that are defined based on a certain election selected in the previous state.
- **Open Poll:** This state is a presentation state in which the User clicks the Open Poll button. The “>>” mark towards the Poll data source asserts that an update will happen in the Poll data source.

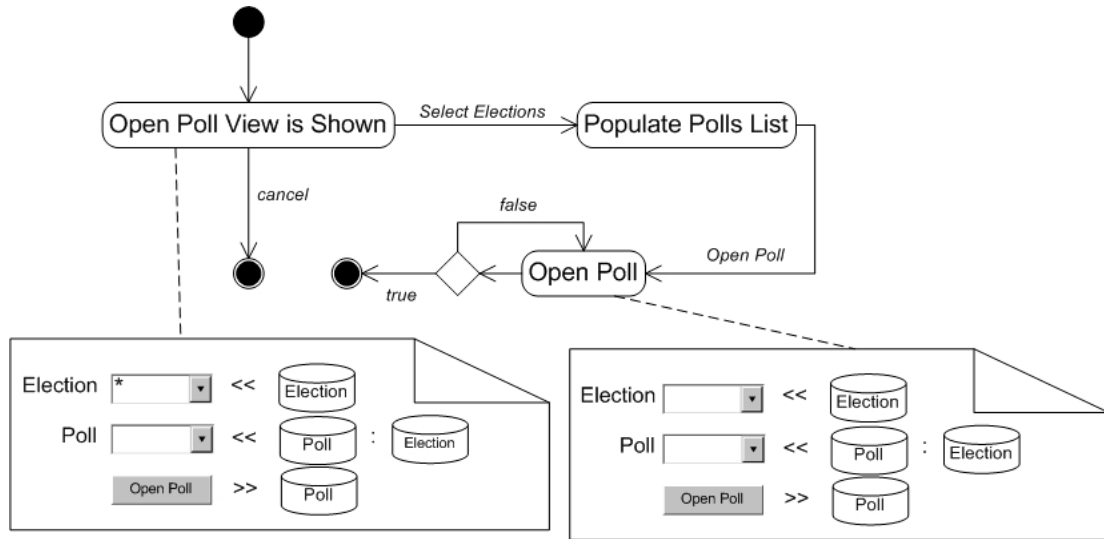


Figure 7 Open Poll Use Case State machine with Presentation UI models

#### 4.1 Generation of the APSM Elements

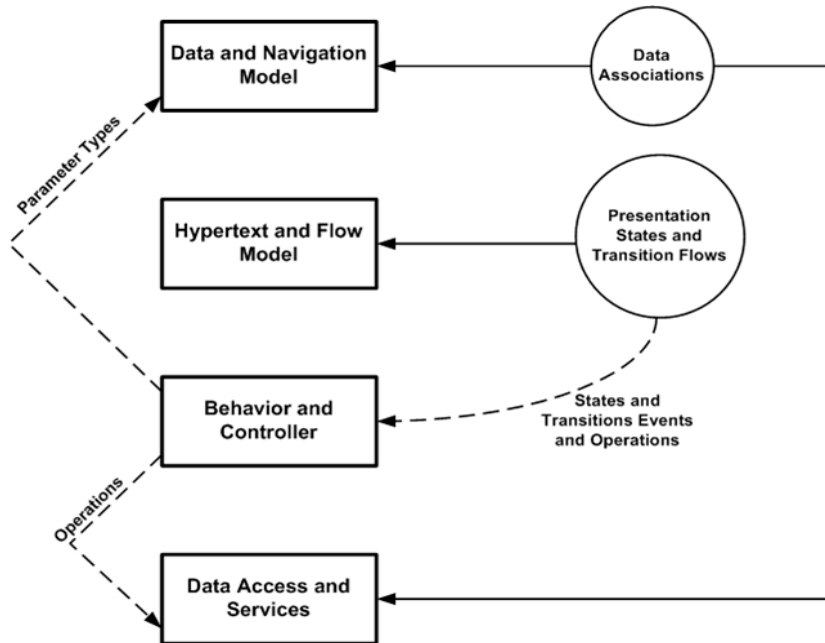


Figure 8 Summary of mappings between a PIM and an APSM. The elements in circles represent the PIM, while the elements in squares are from the APSM.

Figure 8 provides an overall view of the mappings from a PIM to an APSM. The developer defines the state machine in terms of states and transitions, and a UI model attached to presentation states that includes data sources related to every component. The resultant APSM includes: different types of signal/call events that are required to steer the transitions through different web pages, the domain objects and the navigation model that describes the type of associations existing amongst those objects, other objects that act as a collection of objects required to manage information, operations required to access data and control behaviour, and the original state machines and UI models.

The mappings described in Figure 8 are as follow.

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

#### 4.1.1 Data Entities

An entity is generated per data composite associated with the operation trigger of a presentation. UI components contained in the presentation are mapped to entity attributes that belong to the created entity class. For instance, text inputs become *String* attributes and selectable components such as lists and drop-downs are mapped to *Integer* attributes representing the ID of selected elements.

As an example, the state machine in Figure 7 results in the creation of the classes in Figure 9. *PollID* and *electionID* are added because of a drop-down input with the same name as the entities on the page. The association is created because Election drop-down is identified as the default event trigger on the page (with a star symbol). A change in this component will therefore, result in a change on other components in the same page.

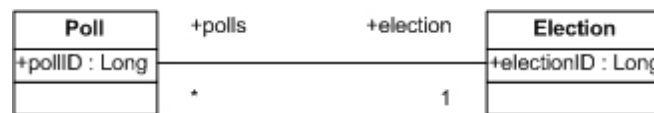


Figure 9 Entity classes generated for the state machine of Figure 7.

#### 4.1.2 Signal Events

Signal events are added to transitions in order to acknowledge the receipt of asynchronous messages. In a web based application, a signal event represents a request by a page for the

execution of a process. The best examples of such events are events fired by submit buttons. In order to generate signal events, every presentation state is examined to verify if it has a submit button (or another component with the same functionality). A signal event is created with the same name as the submit button on the outgoing transition. Parameters to be submitted to the signal event are created according to the set of input fields attached to the presentation.

Signal events are not related only to submit buttons. It often happens that a change in the status of a component other than a submit button results in changes to other components in the same page. Figure 10 shows an example of such situation.

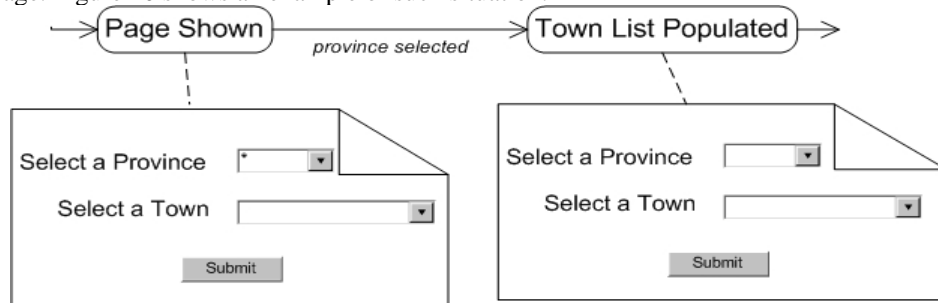


Figure 10 Signal Event created based on the selection event of a drop-down list.

In this example, selecting a province in state *Page Shown* results in the list of towns and villages filled, in state *Town List Populated*. In such a case, the mapping uses the information from the default event holder of the presentation as elaborated in Figure 10, to generate the signal event. When no default event holder is defined, the submit button is taken for that purpose.

The example shown in Figure 7 results in the generation of a signal event with parameter *pollID* on the transition outgoing from state *Open Poll*.

#### 4.1.3 Call Events

Call Events are added to states and transitions. They confirm an operation call and are referred to use-case controllers. A call event is created on a state when an incoming transition to that state carries a signal event. The reception of the event triggers a controller operation, which has the exact same parameters as the signal event.

Two call events are created for the model in Figure 7. One is attached to the *Populate Elections List* state. The second event is normally generated on a state after *Open Poll* state. A new state is automatically added to the model since there is no such state in the model.

#### 4.1.4 Controller Operations

A controller class is created for each use case to grasp front-end operations, perform the required logic treatments and issue calls regarding data services. The most common form of controller operation is the one created upon a signal event (Section 4.1.2). These controller operations are triggered by call events (Section 4.1.3) and use values obtained from the mapping of UI components as parameters. By default, *void* is used as returned type. Other return types are created

for instance when a controller operation performs a check. A typical situation is when an outgoing transition labeled *true* or *false* is considered from a choice. A *boolean* return type is created in such a case.

Controller operations are named according to the following conventions. An operation that aims at controlling a call made by an operation trigger (e.g. submit action) is named by concatenating the individual words making up the operation trigger with every word but the first starting with an upper case character. For instance the controller operation 'openPoll' would be generated to correspond to an operation trigger 'open poll'. In a situation where a controller operation controls a call to an operation that loads items to a select component, the term 'populate' is added as first word.

The following operations are added to the controller in the case of Figure 7:

- *populatePollsList(Election election):void* – This operation uploads from the data store all Polls that are dependent on the provided election parameter.
- *populateElectionsList():void* – It loads all elections from the data store in the list box.
- *openPoll(Poll poll):boolean* – This operation changes the status of a selected poll to open.

We also create an operation for each transition ending in a choice. A natural language description is produced as pseudo-code for that operation. This pseudo-code description is created according to the name of the submit button, the type of the data association used with the submit button and the name of transitions going out of the target choice. The return type is defined based on the state preceding the transition. If the transition comes out of a presentation state with a submit button attached to a data source, a *Boolean* return type is created to indicate the data operation success or failure. In other cases, *Integer* is used as return type in order to give the possibility to handle more than two possible outcomes.

#### 4.1.5 Controllers vs. Services

A controller class is used to control an application according to the behavior defined in a use case [13]. In addition to controller classes, one or more service classes are used to perform operations required for data access. Abstract controllers and services are automatically generated at the APSM level. Different 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. In the same way, a controller is automatically created for every use case. Service operations are basically automatically generated CRUD operations for data composites but the meta-model allows the addition of more complex CRUD operations. Such operations must be defined as a new literal added to the enumerative type

*QueryType*. Also, the required mappings to generate the executable code for the specific platforms must be added to the set of transformations.

Controller operations are automatically created to handle triggers from web-pages. We automatically generate dependencies from a use case controller to the services of all data composites related to that use case. In the same way, dependencies are generated between services and their corresponding data composites.

#### 4.1.6 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. Some examples of when we add page variables to operations aimed at a presentation state are as follows:

- 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 have a page variable carrying a *username*.
- A page variable named *loginAttempts* is created when a login-required state is bound to a maximum number of login attempts.
- A page variable will also be associated to online carts to hold the list of items. This becomes rather a session variable when the page is not login-required.

#### 4.1.7 Domain Objects

Domain objects are inferred from data element associations in UI models. The type of a domain object depends on the type of the element with which it is associated. We use the term *holder* to refer to this element. Data elements are mapped to classes. The attributes of these classes are determined according to the following:

- If the holder is a submit button, the UI components within the same presentation as that submit button are used as source for inferring the attributes. The types of these attributes are defined based on another mapping function that generates an attribute type based on each UI component.
- If the holder is a table, the columns of this table are used as attributes. In practice, tables commonly found in web pages result from composing more than one object but the composite object itself is usually not a member of the database. We do not necessarily deal with concrete database objects but with data objects. These data objects may be composite or single. In the case of single data objects, a data composite would be created including one data entity.
- For other types of UI components the mapping results in a class with no attributes.

Figure 11 shows two examples of a composite data object resulting from a combination of two single data objects. In such cases, we map both the composite and the single objects to classes in the target model with appropriate navigation links. The corresponding controller operation acts on

the composite object but services needed to access the constituent objects are also generated. The first example, presents a case, where the data composite is by itself a data entity whereas the second example presents a case, where the data composite does not belong to the database but is only created to represent a specific view of the User's interest. In both examples, the multiplicity of the *Seat* end is 1 because of the mapping from a single selection component. The multiplicity of the *Candidate* end is *many* in the first example while being 1 in the second one. The former selection results from the fact that *Candidate* supplies a multi selection component. The latter selection is made because of the ':' sign between *Candidate* and *Seat*, which limits the selections of a candidate based on a selected seat.

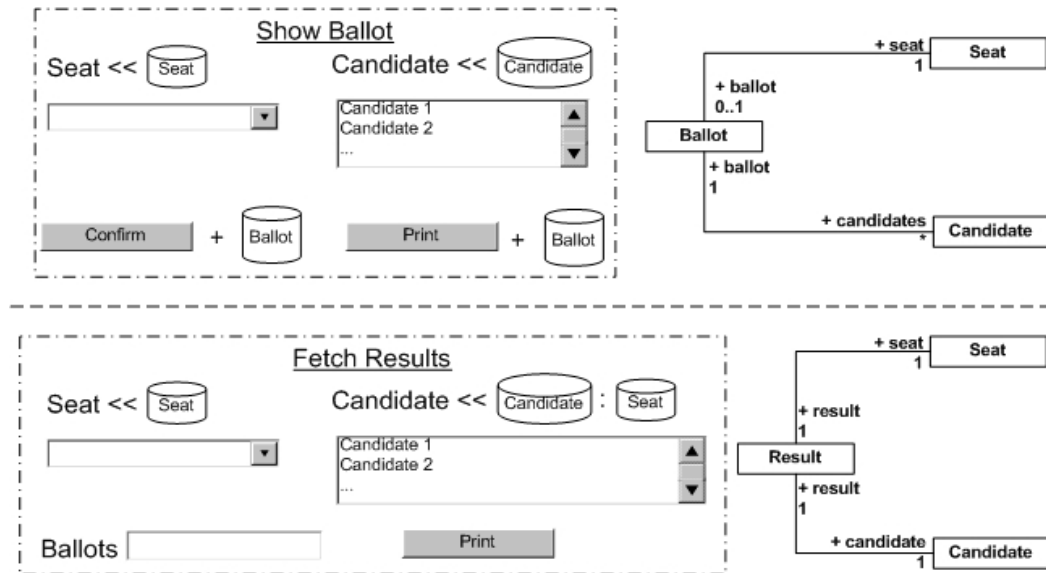


Figure 11 The Mapping of Composite Data Element.

Another type of navigation is formed when mapping class attributes whose types do not belong to the set of primitive data types such as *Long* and *String*. In this case, the type is mapped to a class and an association is generated to support the navigation from the main class to that type class.

Figure 12 illustrates a third type of navigation that is created with respect to composite objects when processing tables. Consider a questionnaire, where the User is provided with a list of questions and a set of possible answers to rate a subject with; the User's answer will be a selected rate for each question. Suppose the results are to be saved in a separate table. In such cases, the object associated with the submit button is created as an aggregation of the objects representing the table columns. The entity *Answer* is created with two one-to-one associations with entities *Question* and *Rate*. The association from *Answer* to *Rate* is one-to-many because the presentation model allows multiple possible rates be assigned to every answer - although the end-user will be only allowed to select one when working with the application.

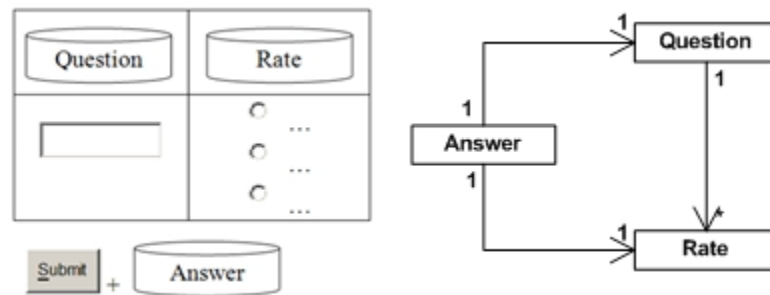


Figure 12 A Composite Objects and Tables.

#### 4.2 Rules for Mapping to a sample Specific Platform

AndroMDA [14] 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 sample platform defined as Java (Programming Language), AndroMDA (Code Generation Framework), MySQL (DataBase Management), DAO (Data Access Mechanism) and Hibernate, ArgoUML and MagicDraw (Modeling Tool) and Struts (UI Framework) - better known as BPM4Struts Cartridge by the AndroMDA community. Hibernate is used as the data access framework. This is required for establishing data transmission amongst different layers. It is based on DAO standards. ArgoUML and MagicDraw are the modeling tools used for refining the resulting PSM. Struts is the UI code framework. Target UI stereotypes and code skeletons are selected from this framework.

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 implying that the stereotyped state represents a web page.
- 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.
    - 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.

The above rules mention very specific details of the platform introduced in this Section. Other elements of the APSM are copied to the SPSM as such.

## 5 A Case Study

We review the approach by applying it to a scenario of the use cases *Login*, *Open Poll* and *Vote*.

### 5.1 Login Use Case

Figure 13 shows a state machine describing use case *Login*. The administrator attempts to log in. The machine verifies the login information and shows a homepage when the login information is

valid. If the information is invalid and the administrator has not reached a maximum of three attempts, he/she is given another chance to login. Otherwise, the account will be locked. Figure 14 presents the UI model of the login state machine in two different sections for states 1 and 4.

Login use case is interesting because of the inclusion of two sequential decision making points that requires creation of appropriate controller operations. Operation *isLoginApproved* is created for the first choice with a *Boolean* return type because the outgoing transitions are labelled true/false. The second choice introduces operation *getNumberOfLoginAttempt* with a return type of *Integer*. Final states are named as other use cases in order to steer the process towards proper use cases in case of success or failure.

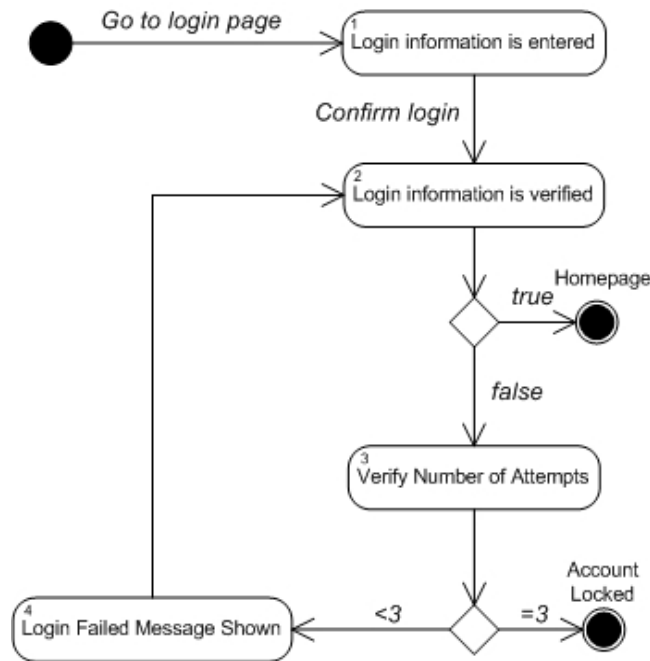


Figure 13 log-in state machine.

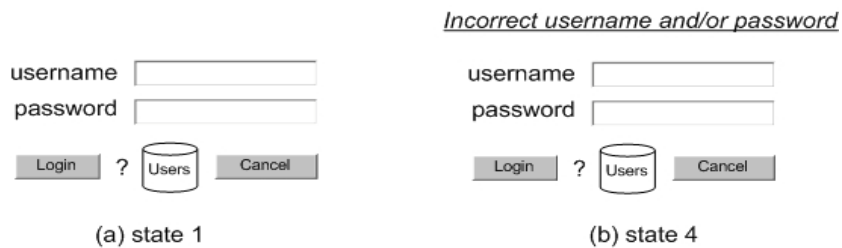


Figure 14 UI Model of login use case

### 5.2 Open Poll Use Case

Another interesting use case is the use case *Open Poll*, which incorporates a situation, where there are more than one operation triggers in one presentation unit. As seen in Figure 15, the User must first select an election in order to populate the list of polls. The presentation of state *Open Poll View is shown* is depicted in Figure 16.

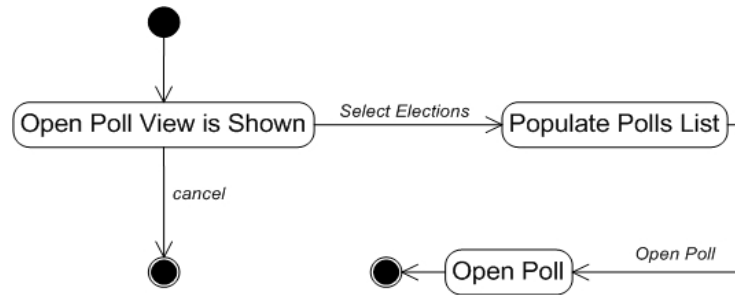


Figure 15 Open Poll State Machine.

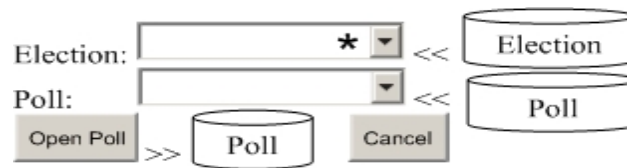


Figure 16 The State *Open Poll View is Shown*

Yet another specific situation in this use case is where the controller is called before the termination of the use case in order to change the status of a poll to *'open'*. Therefore, more than one controller operations are needed. The use case invokes three controller operations: two that are used to load elements from the *election* and *poll* data sources; and a third one to update the *Poll* object in order to change its status to *'open'*. The star placed on the *Election* select component denotes the default event holder of the presentation. The same presentation model is used for the *Populate Polls List* state without the star, which means that the *Open Poll* submit button is used as the default trigger event in that case.

A special characteristic of use case *Open Poll* is the fact that the *Elections* list must be populated first. This is not reflected in Figure 15 because the developers often see the *Open Poll View Shown* as the first step. However, since a call event must be added to the state preceding state *Open Poll View Shown*, the engine adds a new state to the APSM automatically to call the *selectElections* service. In the same way, one state will be added before the final state to process the *Open Poll* event. The final result of the *Open Poll* state machine is shown in Figure 17.

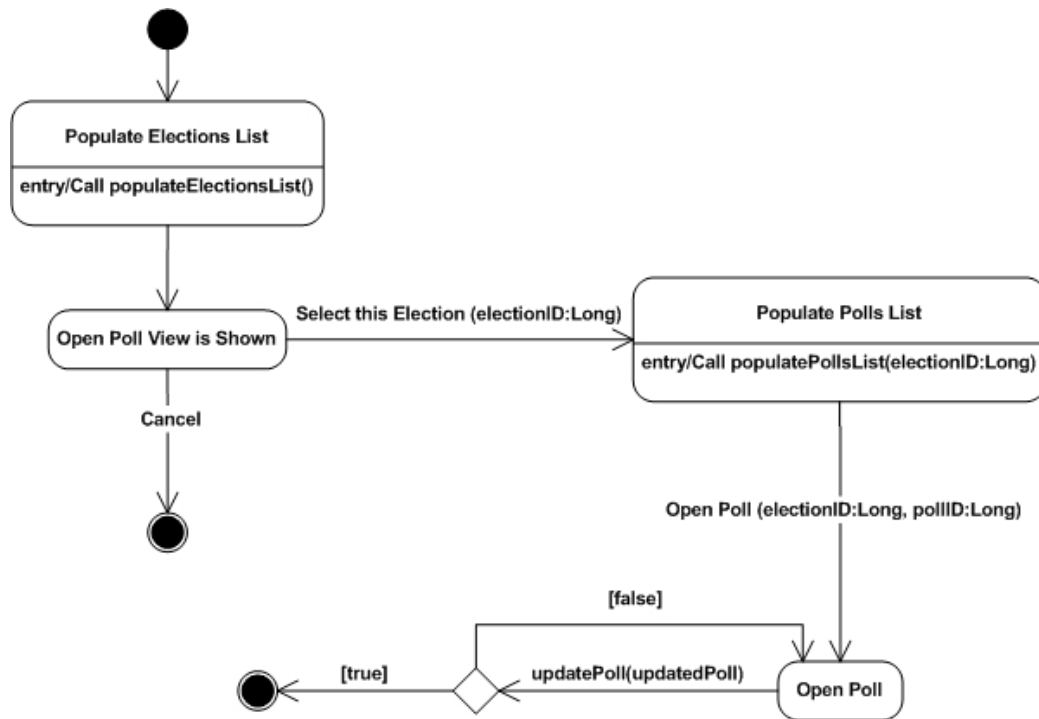


Figure 17 Open Poll State Machine at the APSM level.

### 5.3 Use Case Vote

Use case *Vote* illustrates a couple of more specific situations relevant to the approach. There could be several different scenarios for voting depending on the size and level of an election. In this case, it is assumed that voters log in to a terminal; the system provides a default ballot, in which the voter can see the positions and the names of candidates. The voter fills the ballot out and confirms his/her vote. It is also assumed that only one poll at a time can be opened in a location. A voter can only log in once while a poll is open.

Use case *Vote* also includes a case of composite objects shown in a table. In this case, a domain object, *Ballot* is generated with two attributes/associations:

- *seat:Seat*
- *candidates:Candidate[]*

Figure 18 also shows the fact that the «extend» relation shown in Figure 6 is realized using a state that is named after the extending use case.

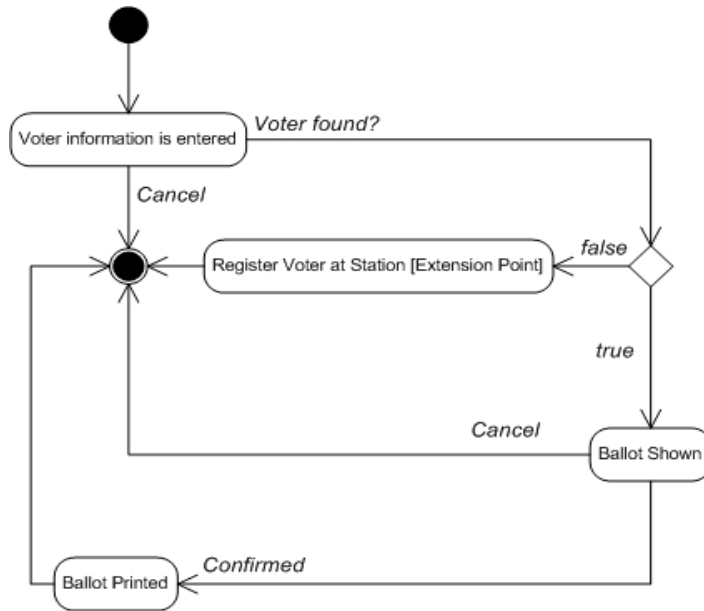


Figure 18 State Machine for use case Vote.

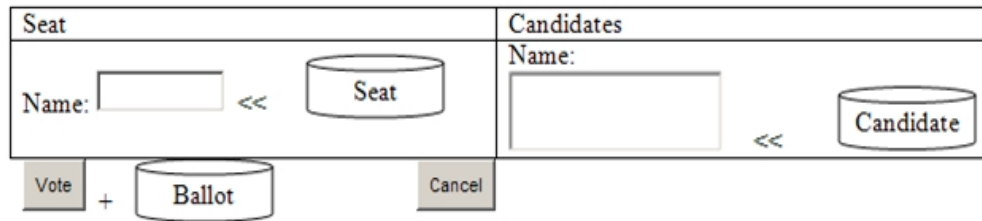


Figure 19 Presentation of a Ballot for voting purposes, *Ballot Shown* state.

The state machine in Figure 18 describes the use case *Vote*. The state machine has a few presentation states. Since we already covered a login use case, we focus on state *Ballot Shown*. In this state, a presentation containing a list of possible seats and candidates is shown. Figure 19 shows this presentation. The ballot is presented using a two-column table. The first column lists the possible seats. The second column shows the list of candidates available per seat.

#### 5.4 Resultant Application

Figure 20 is a class model composed of all classes automatically generated for use cases *Login*, *Open Poll* and *Vote*. In Figure 20, there is a service for each data composite. Controllers may need to be dependent to more than one service depending on how many data items they need access to. Composite aggregations are drawn from data composites to data entities since data composites are composed of entities. Bidirectional associations relate data entities. As the figure shows, controllers and services are operations-only classes while entities are attributes-only ones.

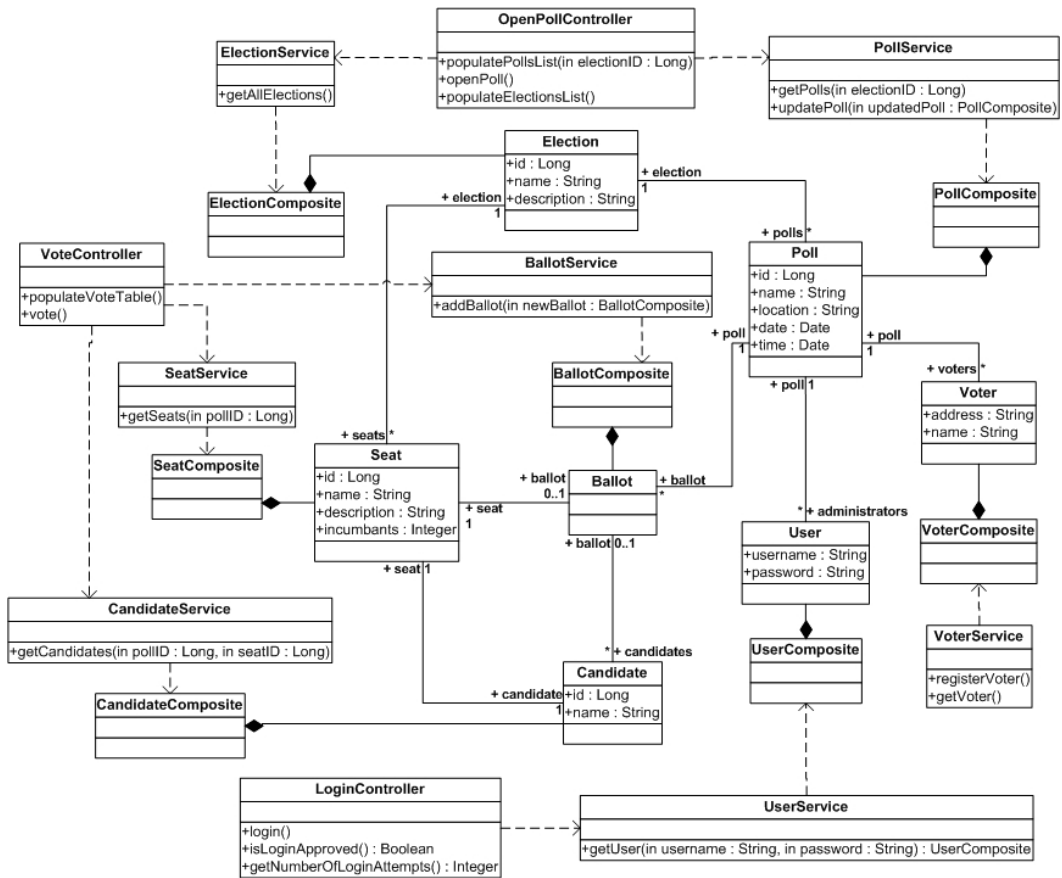


Figure 20 The Automatically Generated Classes at the APSM Level for the use cases *Login*, *Open Poll* and *Vote* of the EMS.

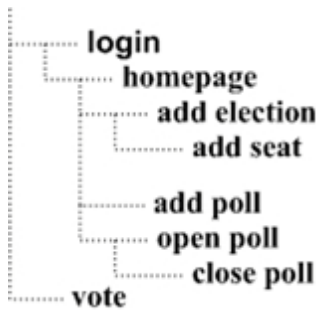


Figure 21 Call structure of the EMS in a happy scenario.

The call structure of the created EMS for a successful scenario is shown in Figure 21.

## 6 Implementation

Our current implementation is based on a set of Eclipse-related tools. The Eclipse Graphical Modeling Framework (GMF) [15] is used to develop editing facilities for our models. We used MediniQVT [16] to define and run QVT relations based on our meta-models. Additionally, a set of Java utilities needed to be developed in order to transform the results obtained from MediniQVT to the file format readable by the target platform (e.g. AndromDA, WebRatio).

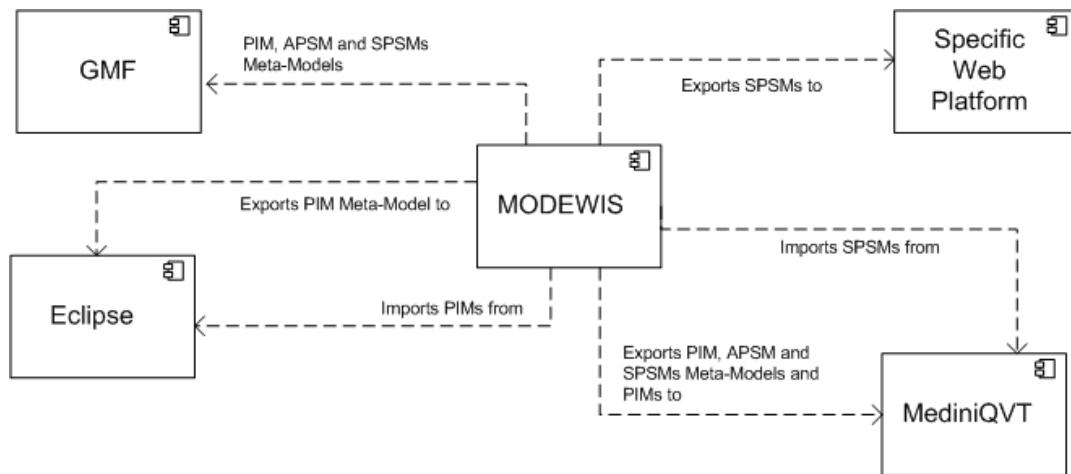


Figure 22 The architecture of MODEWIS.

Figure 22 depicts a conceptual diagram showing the role of different tools for implementing the approach.

- GMF is used for defining the meta-models of the PIM, APSM and SPSMs. The generated models are installed as new eclipse plug-ins.
- Eclipse is the host of the meta-model plug-ins. The input model is created in Eclipse based on the PIM meta-model
- MODEWIS is implemented to assist developers define the input PIM, transform the generated ECORE [17] files from MediniQVT to the target platforms, run the delegated Eclipse-based processes required to copy files, install plug-ins, create meta-models and initiate the modeling activities.

Figure 23 shows a snapshot of four steps in a sample of EMS generated for AndromDA at runtime.

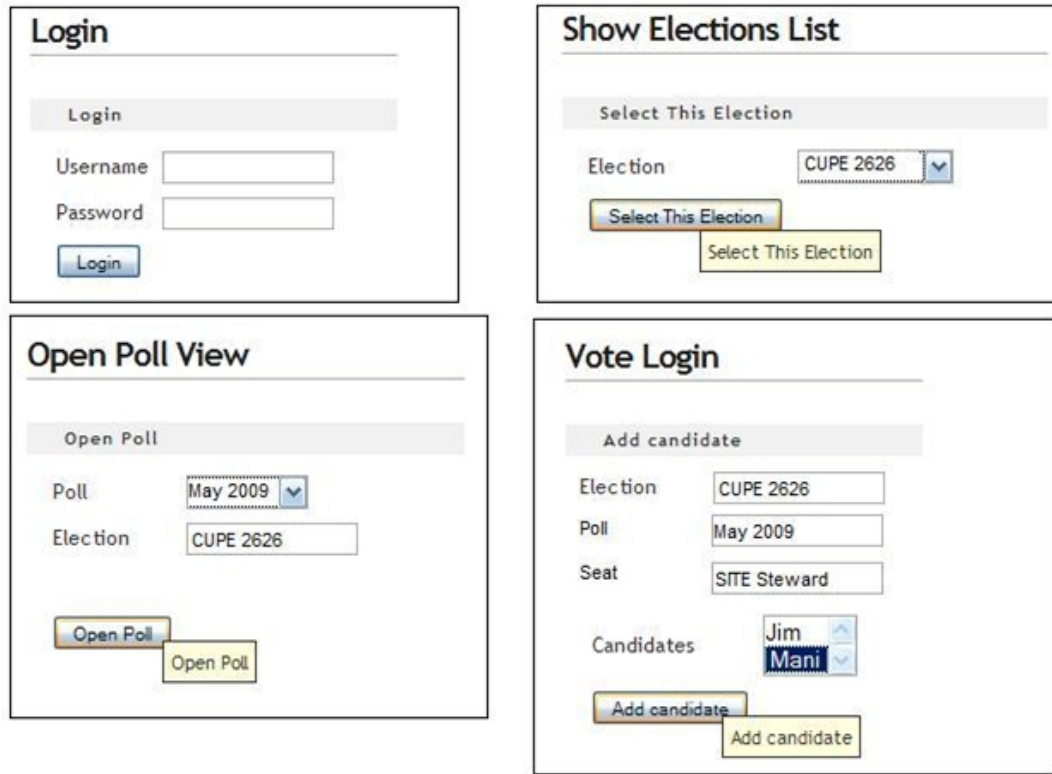


Figure 23 A snapshot of automatically generated EMS executed by AndroMDA.

Figure 24 is a snapshot of MagicDraw that presents the automatically generated code appearing in the documentations of the operation. The code in Figure 24 belongs to the *createUsers* operation of the controller class generated for the use case *Register Candidate*:

- In line 1, a value object is created; the class corresponding to this value object is generated according to the rule 5.e in Section 5.2.
- Lines 3 to 8 assign the required values to the attributes of the created value object; by default the values come from the input fields of the same form.
- In line 10, a call to the corresponding service operation is performed. According to the rule 5.e.iii, there is a dependency from the controller class to the relevant service classes.
- Finally, by default all the create operations carry a *boolean* return value (Section 5.1.4 and Rule 7, Section 5.2); the return value is decided based on the return value from the pertinent service operations in Lines 13-14.



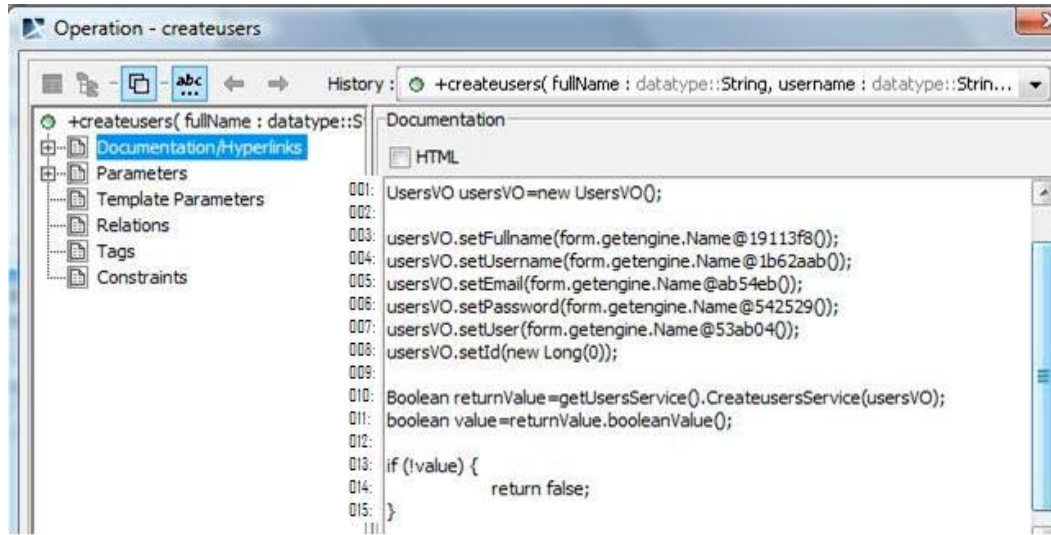


Figure 24 A snapshot of MagicDraw showing the automatically generated code.

## 7 Results

Our approach have been applied to various case studies including: the Election Management System (EMS) [12] already introduced in Section 4, a Team Management System (TMS) [18], which has been used as a course project at the School of Information Technology and Engineering at the University of Ottawa in Fall 2008 for a *Software Design and Architecture* course, and an Account Management System (AMS) that is a project foreseen by a waste/water management company to replace an existing non-web system. In order to evaluate the appropriateness of the approach for Web 2.0 applications, we have also developed a Web 2.0 application [77]. 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, we have recently added another case study that implements a web-based version of a cash/points card for restaurant retailers. This case study covers less use cases compared to our other case studies but implements several situations of operating on more than one instance of one entity in the same page.

As preliminary evaluation, we have compared an automatically generated TMS application with students' projects delivered as part of a course projects. The comparison shows that in most cases the generated application is equal to what students have manually developed in terms of functionality. The result gives an indication that we generate the required functionalities and the essential UI supporting them. In a professional practice, we expect that the generated application would need more customization in terms of web page enhancements and error handling.

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 our

approach. 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. This is what we expected because the focus was not put on creating an enhanced user interface but a minimally functional one. Even with these customization efforts, our approach was still judged as beneficial. The reason is that the existing AMS took one year to be fully deployed while using our approach, the time needed to design all the state machines and presentations was evaluated in days, with a few more days needed to customize the generated application.

The efficiency of the implemented tool also appears reasonable. None of the transformations took longer than 2 minutes for our major case studies: TMS with 15 use cases, AMS with 42 use cases and EMS with 23 use cases. The experiment was done on 2 GHz Intel dual-core processor laptop with 1MB cache and 2 GB RAM.

Finally, the resultant APSMs have been successfully transformed to two different target platforms as well as another web modelling language. One of these target platforms were described in Section 4.2, where an AndroMDA-based profile was targeted. Another target platform is Google Web Toolkit (GWT). As a popular web modelling language, WebML is also a subject of mappings from our meta-model. We have developed mappings to a specific configuration of WebRatio; a tool based on the language WebML.

Currently, the following features are the ones, for which the approach generates the full executable code for the chosen platforms:

- CRUD operations defined in the enumeration QueryType (Figure 1)
- Login state machine and other state machines and classes associated with it
- Classes and operations required to manage online carts, searches and feedbacks

For other features, a partial manual code intervention is required, which varies from case to case.

## **8 Related Work**

In this section, we review various work related to ours. This work is classified in different categories. However, a comparison is only provided for work closely related to ours.

### *8.1 Model-driven work on architectural aspects of web-based application*

Several papers have been published on supporting architectural aspects of web applications in a model-driven sense. A general discussion on basis and necessities of such architecture is provided by Taleb et al. [4]. Hammoudi et al. [19] discuss the formalization of a framework for a typical MDA transformation process. On another note, a UML-based approach to devise the architecture of web-based applications is illustrated by Li et al. [20].

### *8.2 The related work devoted to a specific modeling language*

A number of model-driven web development efforts are based on UWE [21]. A model-driven approach for the semi-automated generation of web-based applications using UWE is introduced by Kraus et al. [22]. This method goes through requirements analysis, conceptual design,

navigation design and presentation design in order to build an application. Automated transformations could be defined to transform models of content, presentation and navigation from model to model and from model to code. The transformation process covers all the levels from CIM to PSM. Examples of such transformations are defined in the series of the UWE-related works. Kraus et al's method is an approach to generate web-based applications that provide detailed mechanisms to define web applications in a model-driven way. The main difference to ours appears to be the purpose. As a part of the group of UWE-based approaches, Kraus et al's effort is dedicated to preparing a framework based on which developers can generate model-driven applications. For this purpose, they mention an example of transformations created based on their method. Our approach is instead focused on (semi)automating parts of the process; thus we cover a smaller scope in a rather automated manner. We do not tend to provide a base method on top of which, others can build their own ones; instead our method could be seen as a sample method of what model-driven web engineering approaches such as UWE might be able to generate.

Brambilla et al. [23] introduce an approach for web development using WebRatio [24] as a tool and WebML [25] as a modeling language. The approach guides the developer through a model-driven method to define a web-based application in terms of the following: Data modeling, Hypertext modeling, Personalization to give different Users different viewpoints, Presentation to add the look-and-feel, Integrating business processes and Web services. The result is a PSM that is automatically mapped to executable code. Brambilla et al's approach focuses on the automation of the PSM-to-Code transformation while providing detailed mechanisms for developing the PSM. Our approach is instead centered on the PIM-to-PSM transformation and leaves code generation to other tools. Lowe and Tongrunrojana [26] suggest an extension to WebML in order to bridge high-level business models with lower-level web models. The authors use the same language at another level to define the architectural aspects of the flow of information [27]. They also provided a UML-compliant version of their suggested language to describe information flows [28].

In [29], the authors suggest a model-driven method for the generation of web based applications using the Object Oriented model for Hypertext Data-centric Modeling approach (OOHDM) [30] - has its roots in HDM (Hypermedia Design Model). The method covers a software engineering process from use cases to implementation. Use cases are modeled using interaction diagrams. There is also a spot for navigational modeling using state charts as well as conceptual modeling by class diagrams. The main difference between this approach and ours is our coverage to automation especially in terms of data access and UI modeling.

Another approach based on OOHDM is published by Schmid and Donnerhak [31]. This approach results in two different PSMs: conceptual and navigational. It has the advantage that different independent technologies for presentation and logic can be chosen. However, the approach only covers navigational transformations. A semi-formalized language is used to generate the PSM. The target platform is a servlet-based web application. Besides not covering the conceptual parts of the application, the main difference between this approach and ours is the fact that the UI model is not considered as part of the requirements. Generally speaking, the approach is not concerned with requirements. Yet another approach based on OO-H - as a successor of OOHDM - is presented by Gómez [32].

### *8.3 Model-driven approaches to UI-related features of web applications*

A group of researchers have worked on UI-related aspects of web applications. Wu et al. [33] describe a method to generate user interface code following MDA transformations and the Model View Controller (MVC) pattern. The method spans the gap from requirements to code for a user interface model by transforming boundary objects resulting from robustness analysis [34] to JSP pages [35]. The authors provide a framework that starts with use case modeling and activity diagrams. Then they perform a robustness analysis to categorize the participating objects. Finally, JSP pages are built from the UML models according to transformation rules. Unlike Wu et al.'s work, our approach covers the generation of code for the whole software system not only the user interface part. Although we select certain platforms to implement our method, the method itself is platform-independent. Other approaches for generating web UIs by model-driven approaches exist. Costa et al. [36] for example, present an MDA-compliant method to devise a step in user interface design for web applications in accordance with UML version of ConcurTaskTree (CTT). De Souza and de Barros [37] provide another method using JSFs for generating web UI as an eclipse plug-in. Sukaviriya et al.'s work [38] is a process framework for HCI lifecycle of web applications based on iterative/incremental approach. WSDM [39] may also be categorized in this group that suggests defining the web system based on its users groups. An interesting wizard-based approach to derive the user-interface model required to support existing domain models is suggested by Stocq and Vanderdonck [40]. A technique for classifying distributed user interfaces is reported by Demeure [41] using a reference model. Sousa et al.'s work [42] is an effort to resolve problems that exist with present tools and techniques for model-driven user-interface development by employing a strategy that aims at usability issues as the top-most priority.

### *8.4 Partial solutions to automated web engineering*

There are other model-driven approaches to web engineering that approach the solution not in whole but partially. For example, a model-driven methodology for composing web applications using model-driven techniques is published by Kateros et al. [43]. The method uses model-driven references and transformations to configure web applications. Vara et al. [44] provide a model-driven method for the generation of object-oriented databases to be used in a web information systems method named MIDAS [45]. A well-defined approach to support web information systems with multiple data sources is presented by Vdovjak and Houben [46]. Another data-related work is presented by Whitehead et al. [47], where authors describe an approach to bring repositories to the hypertext level so that they could be visualized using the web. Another series of work that discusses the presentation of data and document in hypertext level is provided by Bieber and Wang [48], Wnag and Bieber [49].

### *8.5 Model-driven efforts recognizing the notion of abstract web or notions alike*

He et al. [6] recognize the same problem and suggest a similar approach as ours. However, their discussion remains at a conceptual level. Ubiquitous Web Application (UWA) [50] is another example of a general framework for defining an abstract web application. However this approach is highly abstract. UWA is suitable as a reference framework that one might use to build a method or a model. Thus, in order to use UWA, one must first define the platform or abstract platform

based on UWA framework. As a result UWA itself may not be directly used in practical approaches. Muller et al.'s approach [7] is another closely related approach, in which the authors separate the PSM into platform-dependent and technology dependent PSMs. The two are comparable to the notion of APSM and SPSMs used in this paper. Muller et al.'s approach is however focused on the transformation from APSM to SPSM for one specific platform, while our focus is on PIM to APSM transformation. A J2EE framework for model-driven development of web applications based on a universal web model with a focus on authentication features is also found in Sakowicz et al.'s method [5].

#### *8.6 Model-driven methods that take legacy code into consideration*

An independent track of research is devoted to re-engineering/re-use of legacy web-based systems. A method to reverse engineering UML class models from web pages is provided by Pu et al. [51]. Another method for tracking customized manual code added to automatically generated applications is provided by Cichetti et al. [52]. Nguyen and Chun [53] present an effort to synchronize interaction diagrams representing use cases with legacy code using model-driven techniques.

#### *8.7 Model-driven requirements engineering for web applications*

A recent approach to address the problem of modeling requirements of web information systems is published by Molina et al. [54]. Cuaresma and Aragón's method [55] is a navigational-based method to model requirements of a web-based system which will eventually be used in a model-driven context. The method is supported by a tool named NDT [56]. SOHDM [57] and [58] is one of the first methods proposed to deal with web requirements. SOHDM contributes scenarios as a technique to elicit requirements [59]. Another approach, which is closely related to UWE is described by Koch et al. [60]. An approach that generates a full web application from requirements is suggested by Liang et al. [61]. This approach re-uses two different toolkits for requirements definition in natural language and application generation but the main contribution is to convert the models from the former toolkit to the latter one.

#### *8.8 Model-driven environments for building web-based applications*

Guidelines, frameworks and principals required to set up a model-driven development environments are addressed by a number of researchers. Such environments are in some cases specific to certain tools and techniques. One example that supports NDT tool is Escalona et al.'s approach [62]. The NDT suite provides a set of tool and techniques for requirements elicitation for web applications with an emphasis on the flow of events and transitions among different user views. Other examples discuss more general parameters that could be used towards all model-driven development methods. Such a conceptual framework for MDA-based software development environments is described by Pastor and Molina [63]. Since an ultimate goal of our method could be the generation of an MDA-based environment, this work provides a useful point of reference.

### 8.9 *Methods using domain-specific languages*

Most of the work on UWE, WebML and W2000 could be considered in this category because those languages are DSLs. As another example, a DSL for building web dialogs using model-driven techniques is suggested by Freudenstein et al. [64]. Another method that combines DSLs and model-driven development for rapid prototyping of web applications is presented by Nunes and Shwabe [65].

### 8.10 *Analysis*

The following features of our work have not been fully addressed by other work:

- Data access generation: Even the papers addressing data-centric or information-based web applications - such as [66-68] 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, which will be discussed in more detail.
- Introducing the notion of APSM and SPSMs is another distinguishing aspect of our approach. Existing transformations are usually defined towards a PSM. Exceptions such as [6] and [50] end up with models that are very general and too abstract. As a result, most of the research lead to approaches that are either hard to map to specific platforms or hard to adapt with other platforms. Our approach distinguishes APSM and SPSM so that the models and the mappings could be flexible in terms of adaptability to web platforms.
- As described in [69], most of the existing models and methods are affected by name-based mapping rules that can cause ambiguities with regard to the separation of concerns. Supplying semantic relationships to replace the name-based associations has been amongst our intentions for extending an existing model. For example, although name-based rules play a key role in defining the association between a *UIComponent* and a *DataComposite*, the association would be concrete in the instance model according to our meta-model. This concrete association instance between a *UIComponent* and a *DataComposite* provides the same type of support as suggested by Cichetti and Di Ruscio [69] in terms of an association from *DataCompositionWLink* and *CompositionWElement*.
- Traditional web modeling languages including WebML are not based on UML/MOF according to [70]. There has been a number of efforts to adapt WebML with MDA/MOF/UML family such as [70-73]. These suggest that it is an advantage to design a model that is UML-based. Our model is an extension to another UML-based model, which has been designed following the UML extension mechanism.
- Our mappings are extensively supported by sets of QVT relations that transform PIM to APSM and APSM to example SPSMs. Appendix 1 presents a list of such relations.
- Our approach pays specific attention to several features of web information systems such as authentication and session mechanisms, which are not widely handled in other approaches.

- Web 2.0 [76] is another important feature of our work. Valverde and Pasotr (2009) describe a meta-model to specify technical details of a Web 2.0 platform and Hernández et al. (2010), explain an approach towards defining a meta-model for the definition of the conceptual aspects of web 2.0 as a venue for social networking. Two main features of our model that help supporting Web 2.0 are the facts that 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 may accept feedback, which supports the interaction of contents as required by Web 2.0.
- Finally, our approach closely relates requirements to lower-level models and hence increases the chance of developing applications in accordance with requirements.
  - Use cases are systematically addressed by our approach and their behaviour is strongly modelled using state machines. Several features are provided to suggest mappings from different use case structures such as inclusion/extension, use case steps and scenario use cases directly to state machines. Hence, the resultant APSM state machines inherit the same type of relationships that exist amongst use cases.
  - We see the 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. in [33]. 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.

Amongst the mentioned related work, there are approaches that have similarities with ours but none has all the characteristics that we propose. For example, Wu et al's proposes a set of detailed transformation rules but only covers the presentation layer. UWE-based approaches result in an abstract web-specific model that could be mapped to any platform but they do not provide detailed transformation rules. Instead they focus on the UWE meta-model and how to use it. This is also true about the approaches using OOHDM. Automation is also a feature missing from most of the related work. Furthermore, much of the existing work covers the topic of data modelling and navigation modelling. However, data modelling is different from the data access mechanism. The latter is missing from most of the related work. Modelling the data access through the automated generation of classes and operations required to retrieve and store data is another major concern of our approach.

We can conclude that WebML-based approaches present the most comprehensive way of data modelling for web applications. Further, WebRatio 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 approach, which is the PIM-APSM transformation, is not dependent on the database. It is also worth remembering

that WebML is originally a domain-specific language (DSL) rather than a UML/MOF-based model. While being a DSL is not a disadvantage by itself, it certainly is an advantage for a model to be compatible with UML. Another difference is that data modelling in WebRatio as the most famous WebML tool is largely manual while our approach automatically discovers the data model.

## 9 Conclusion

We have presented an approach for automated generation of web information systems. Our approach is mainly based on the idea of using an abstract platform as the basis of transformations. This has enabled our approach to handle transformations from PIM to multiple PSMs by inserting an intermediate level, the APSM. The invention of APSM increases the level of re-usability of models and mappings. We have paid specific attention to defining requirements and mapping them to lower-level models. Our approach has achieved the goals of specifying the operations, classes and associations to supply required data throughout the application layers and generating the corresponding code for specific platforms. All these have been established while maintaining a flexible application architecture in terms of the number and configuration of its layers and objects.

We accept state machines and UI models attached to presentation states as input. That input constitutes a PIM that provides a high-level abstract description of a web application. Presentation states represent web pages and the transitions represent the flow between pages. The data associations are used to build up the data model as well as the data access operations. The events attached to states and transitions are used to generate required behavior to control the web application. The application generated in the first stage is an abstract web application that may be transformed to certain specific platforms in the second stage. Our approach has a particularity that it assumes certain knowledge of the data model when devising the PIM. We found this to be a reasonable assumption in the context of web-based information systems, where the focus is generally on providing a web access to existing information sources. Even when the sources do not exist, our assumption is reasonable because the knowledge of those resources is known to developers to some extent based on the requirements

In our future work, we plan to improve the usability of our prototype tool. An ultimate objective is to change the appearance of environment so that we could move from a state machine-based input form towards a graphical flow form so that a non-expert User can use the tool for generating customized web information systems. We will also explore the option of using textual use case descriptions by integrating MODEWIS with UCED [74] a tool developed in a previous work that automatically generates a state model from use cases in text form. The integration would provide an extension of the approach to the Computer Independent Model (CIM). Further work will also enhance the relations and the algorithms as well as extend the approach to cover additional UI elements such as multi-media components. We will also continue our experiments with different platforms to support more platform-specific mappings. We will specially focus on platforms for which specific mappings have already been published. The invention of new QVT engines such as [75] and updates to existing tools such as [16] is convincing enough to re-use those tools instead of developing the custom QVT engine as another part of the future work.



## References

1. Shklar, L. Rosen, R. Web Application Architecture : Principles, Protocols, and Practices. Chichester, England ; Hoboken, NJ : John Wiley, c2003.
2. Meliá, S. Gómez, J. Applying Transformations to Model Driven Development of Web Applications. In: First International Workshop on Best Practices of UML (BP-UML 2005), In : ER Conference 2005. pp. 63-73
3. Ceri, S. Fraternali, P. and Bongio, A. Web Modeling Language (WebML): a modeling language for designing Web sites. Computer Networks 33 (1-6) 2000. pp. 137-157
4. Taleb, M. Seffah, A. Abran, A. Model-Driven Architecture for Web Applications. In: Human-Computer Interaction (1) 2007. pp. 1198-1205
5. Sakowicz, B. Murlewski, J. Labus, A. Napieralski, A. JWay - Model-Driven J2EE Application Framework. In: Proceedings of the International Conference of Mixed Design of Integrated Circuits and Systems, 2007. pp. 703-706
6. He, C. Tu, W. and He, K. Role Based Platform Independent Web Application Modeling. In: Proceedings of the Sixth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'05). PP. 411-415
7. Muller, P. A. Studer, P. Fondement, F. and B'ezivin, J. Platform independent Web application modeling and development with Netsilon. In: Software and System Modeling 4(4) 2005. pp. 424-442
8. OMG, MDA Guide Version 1.0.1, 12-06-2003
9. Microsoft .Net Framework, <http://www.microsoft.com/NET/>, September 2009
10. Epner, M. Poor project management number-one problem of outsourced eprojects. In: Research Briefs, Cutter Consortium (2000). Available from: <http://www.cutter.com/research/2000/crb001107.html>
11. 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, 2007
12. Lethbridge, T. C. and Laganière R. Object-oriented software engineering : practical software development using UML and Java, London : McGraw-Hill, c2001
13. Larman, C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Prentice Hall PTR. 2005.
14. AndroMDA, [www.andromda.org](http://www.andromda.org), 15-02-2007
15. Graphical Modeling Framework, [www.eclipse.org/modeling/gmf](http://www.eclipse.org/modeling/gmf), April 2009

- 142 *Model-Driven Web Development for Multiple Platforms*
16. mediniQVT – Trac, <http://projects.ikv.de/qvt>, 3 May 2008
  17. Eclipse Modeling - EMF - Home, [www.eclipse.org/modeling/emf](http://www.eclipse.org/modeling/emf), January 2009
  18. Stéphane Sotèg Somé's homepage, <http://www.site.uottawa.ca/~ssome/>, Fall 2008.
  19. Hammoudi, S. Alouini, W. Lopes, D. Towards a Semi-Automatic Transformation Process. In: MDA - Architecture and Methodology. In: International Conference of Enterprise Information Systems (3-2) 2008: 416-425
  20. Li, J. Chen, J. Chen, P. Modeling Web application architecture with UML. In: Proceedings of the 36th International Conference on Technology of Object-Oriented Languages and Systems, 2000. pp. 265-274
  21. UWE – UML-based Web Engineering, [www.pst.informatik.uni-muenchen.de/projekte/uwe/](http://www.pst.informatik.uni-muenchen.de/projekte/uwe/), 3-08-2008
  22. Kraus, A. Knapp, A. and Koch, N. Model-Driven Generation of Web Applications in UWE. In: Proceedings of the 3rd International Workshop on Model-Driven Web Engineering, In: CEUR-WS, Vol 261, 2007
  23. Brambilla, M. Comai, S. Fraternali, P. and Matera, M. Designing Web Applications with WebML and WebRatio. In: Web Engineering: Modelling and Implementing Web Applications. Gustavo Rossi, Oscar Pastor, Daniel Schwabe and Luis Olsina. 2007
  24. WebRatio, [www.webratio.com](http://www.webratio.com), 6-5-2008
  25. WebML, [www.webml.org](http://www.webml.org), May 5, 2008
  26. Lowe, D., Tongrunrojana, R. WebML+: a Web modeling language for modeling architectural-level information flows. In: Proceedings of The Twelfth International World Wide Web Conference, 2003. pp. 17-24
  27. Tongrunrojana, R. Lowe, D. WIED: A Web Modelling Language for Modelling Architectural-Level Information Flows. In: Journal of Digital Information, Vol 5, No 2 2004.
  28. Lowe, D. and Tongrunrojana, R. Web Information Exchange Diagrams for UML. In: Proceedings of Web Information Systems – WISE 2004. pp 29-40.
  29. Rossi, G. Schwabe, D. Model-Based Web Application Development. In: Web Engineering: Theory and Practice of Metrics and Measurement for Web Development. E. Mendes and N. Mosley, Springer, 2006. pp. 303-333
  30. The Object-Oriented Hypermedia Design Model (OOHDM), [www.telemidia.puc-rio.br/oohdm/oohdm.html](http://www.telemidia.puc-rio.br/oohdm/oohdm.html), May 5th, 2008

31. Schmid, H. A. and Donnerhak, O. The PIM to Servlet-Based PSM Transformation with OOHDMDA. In: Proceedings of Workshop on Model-driven Web Engineering (MDWE 2005), 2005
32. Gómez, J. Model-Driven Web Development with VisualWADE. In: International Conference on Web Engineering 2004. pp. 611-612
33. Wu, J. H. Shin, S. S. Chien, J. L. Chao, W. S. and Hsieh, M. C. An Extended MDA Method for User Interface Modeling and Transformation. In: The 15th European Conference on Information Systems 2007. pp 1632-1641
34. Rosenberg, D. and Stephens, M. Use case driven object modeling with UML : theory and practice, Apress Publishers (2007)
35. Java Server Pages Technology, [java.sun.com/products/jsp](http://java.sun.com/products/jsp), June 23, 2004
36. Costa, D. Nóbrega, L. and Nunes, N. J. An MDA Approach for Generating Web Interfaces with UML ConcurTaskTrees and Canonical Abstract Prototypes. In Book: Task Models and Diagrams for Users Interface Design, Springer Berlin / Heidelberg 2007. pp. 137-152
37. de Souza, R. A. C. de Barros, R. S. M. A Model-Driven Method for the Development of Web Applications User Interaction Layer. In: TASE 2008. pp. 91-98
38. Sukaviriya, N. Sinha, V. Ramachandra, T. Mani, S. Model-Driven Approach for Managing Human Interface Design Life Cycle. In: MoDELS 2007. pp. 226-240
39. De Troyer, O. Leune, C. WSDM: a user-centered design method for web sites. In: Computer networks and ISDN systems. In: 7th international world wide web conference. 1998. pp. 85–94.
40. Stocq, J. and Vanderdonckt, J. A domain model-driven approach for producing user interfaces to multi-platform information systems. in: The Proceedings of the working conference on Advanced visual interfaces, 2004. pp. 395-398
41. Demeure, A. Calvary, G. Sottet, J. S. and Vanderdonkt, J. A reference model for distributed user interfaces. In: The Proceedings of the 4th international workshop on Task models and diagrams. 2005. pp 79-86.
42. Sousa, K. Mendonça, H. and Vanderdonck, J. Towards Method Engineering of Model-Driven User Interface Development. In: Task Models and Diagrams for User Interface Design 2007. pp 112-125.
43. Kateros, D. A. Kapitsaki, G. M. Tselikas, N. D. Venieris, I. S. A Methodology for Model-Driven Web Application Composition. In: IEEE International Conference on Services Computing, 2008. Volume 2, Issue 7-11, 2008. pp. 489-492

44. Vara, J. M. Vela, B. Cavero, J. M. Marcos, E. In: Model transformation for object-relational database development. In: Proceedings of the 2007 ACM symposium on Applied computing table of contents. pp. 1012 – 1019
45. Cáceres, P., Marcos, E., Vela, B.: A MDA-Based Approach for Web Information System Development. In: Workshop in Software Model Engineering (WiSME), In: Proceedings of UML 2003.
46. Vdovjak, R. and Houben, G. J. A Model-Driven Approach for Designing Distributed Web Information Systems. In: Web Engineering, 2005. pp. 453-464
47. Whitehead Jr., E. J., Ge, G. and Pan, K. Automatic generation of hypertext system repositories: a model driven approach. In: The Proceedings of the fifteenth ACM conference on Hypertext and hypermedia, 2004. pp 205-214.
48. Bieber, J. W. Wang, M. LHM: a logic-based hypertext data model for integrating hypertext and information systems. In: The Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences, 1995. Vol. III. pp. 350-359
49. Wang, J. and Bieber, M. GHMI: a general hypertext data model supporting integration of hypertext and information systems. In: The Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences, 1996. Volume: 2, pp. 47-56.
50. UWA Consortium, Ubiquitous Web Applications. In: Proceedings of the eBusiness and eWork Conference 2002, (e2002: October 16-18 2002, Prague, Czech Republic) (2002)
51. Pu, J. Yang, H. Xu, B. Xu, L. Chu, W.C.C. Combining MDE and UML to Reverse Engineer Web-Based Legacy Systems. In: COMPSAC 2008. pp. 718-725
52. Cicchetti, A. Di Ruscio, D. Di Salle, A. Software customization in model driven development of web applications. In: SAC 2007. pp. 1025-1030
53. Nguyen, P. Chun, R. Model Driven Development with Interactive Use Cases and UML Models. In :Software Engineering Research and Practice 2006. pp. 534-540
54. Molina, F. Pardillo, J. Ambrosio, J. Álvarez, T. Modelling Web-Based Systems Requirements Using WRM. In: WISE Workshops 2008. pp. 122-131
55. Escalona Cuaresma, M. J. Aragón, G.: NDT. A Model-Driven Approach for Web Requirements. In: IEEE Transactions on Software Engineering 34(3) 2008. pp. 377-390
56. Escalona, M.J. Torres, J. Mejías, M. and Reina, A.M. NDT-Tool: A Tool Case to Deal with Requirements in Web Information Systems. In: Proceedings of the Fourth Int'l Conf. Web Eng. 2003. pp. 212-213
57. Lee, H. Lee, C. Yoo, C. A scenario-based object-oriented methodology for developing hypermedia information systems. In: 31st IEEE Annual conference on systems science. Sprague R, 1998. pp. 121–38

58. Suh, W. Lee, H. A methodology for building content-oriented hypermedia systems. In: *Journal of Syst Software* 2001, 56. pp. 115–31.
59. Weidenhaupt, K. Pohl, K. Jake, M. Haumer, P. Scenarios in system development: current practice. In: *IEEE Software* 1998;2. pp. 34–45.
60. Koch, N. Zhang, G. and Escalona, M. J. Model transformations from requirements to web system design. In: *Proceedings of the 6th international conference on Web engineering, 2006*. pp. 281 – 288
61. Liang, X. Kop, C. Ginige, A. and Mayr, H. C. Turning Concepts into Reality - Bridging Requirements Engineering and Model-Driven Generation of Web-Applications In: Joaquim Filipe, Markus Helfert, Boris Shishkov (Eds.), *Proceedings of the Second International Conference on Software and Data Technologies (ICSOFT 2007), INSTICC Press, Barcelona, Spain, 2007*. pp. 109 – 116.
62. Escalona, M.J. Morero, F. Parra, C.L. Nieto, J. P\erez, F. Martín, F. Llergo A. and Guti\errez J.J. A Practical Environment to Apply Model Driven Web Engineering. In: *Information Systems Development*. Vol. 1. 2009. pp. 249-257
63. Pastor O. and Molina J. C., *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling*, Springer (2007)
64. Freudenstein , P. Nussbaumer , M. Allerding , F. Gaedke, M. A domain-specific language for the model-driven construction of advanced web-based dialogs. In: *Proceeding of the 17th international conference on World Wide Web, 2008*. pp. 1069-1070
65. Nunes, D. A. and Schwabe, D. Rapid prototyping of web applications combining domain specific languages and model driven design. In: *ICWE 2006*. pp. 153-160
66. Ceri, S. Fraternali, P. Bongio, A. Brambilla, M. Comai, S. Matera, M.. *Designing Data-Intensive Web Applications*. Morgan Kaufmann. 2006
67. Nikolaidou, M. and Anagnostopoulos, D. A Systematic Approach for Configuring Web-Based Information Systems. In the *Journal of Distributed and Parallel Databases*, Springer Netherlands, 17(3) 2005. pp. 267-290
68. 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, 2006. pp. 335-408
69. Cicchetti, A. Di Ruscio, D. Decoupling Web Application Concerns through Weaving Operations. In: *Science of Computer Programming* 70(1) 2008. pp. 62-86
70. Brambilla, M. Fraternali, P. Tisi, M. A Metamodel Transformation Framework for the Migration of WebML Models to MDA. In: *4th Int. Workshop on Model-Driven Web Engineering (MDWE 2008)*. N. Koch, G.-J. Houben, A. Vallecillo (Eds.). In: *CEUR Proceedings*, volume 389. pp. 91-105.

71. Moreno, N. Fraternali, P. Vallecillo, A. A UML 2.0 profile for WebML modeling. In: Workshop proceedings of the sixth international conference on Web engineering, Second international workshop on model driven web engineering (MDWE'06) 2006
72. Moreno, N. Fraternali, P. Vallecillo, A. WebML modeling in UML. In: IET Software Journal (2007). pp. 67-80
73. 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. In: IET SOFTWARE, 1-3, pp. 81 - 97
74. Use Case Editor, [http://www.site.uottawa.ca/~ssome/Use\\_Case\\_Editor\\_UCEd.html](http://www.site.uottawa.ca/~ssome/Use_Case_Editor_UCEd.html), Feb 2006
75. Declarative QVT, Quick Start, [http://www.eclipse.org/m2m/dqvt/quick\\_start.pdf](http://www.eclipse.org/m2m/dqvt/quick_start.pdf), March 23,2009
76. Tim O'Reilly (2005-09-30). "What Is Web 2.0". O'Reilly Network. Available from: [www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html](http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html). Retrieved 2006-08-06.
77. Dilmaj, [http://sokhangozaar.appspot.com/?locale=en\\_US#](http://sokhangozaar.appspot.com/?locale=en_US#), July 2010

## Appendix 1: QVT Relations

```
top relation application_application {
  ucName:String; applicationName : String;

  checkonly domain wPIM application1 : wpim::Application {
    name = applicationName,
    useCases=usecase1:wpim::UseCase{
      name=ucName,application=application1}
  };

  enforce domain aPSM application2 : apsm::Application {
    name = applicationName,
    useCases=usecase2:apsm::UseCase{
      name=ucName,application=application2}
  };
  where {usecase_usecase(usecase1, usecase2);}
}

top relation application_siteMap {
  applicationName:String;

  checkonly domain wPIM application1:wpim::Application {
    name=applicationName,
    useCases=usecase1:wpim::UseCase {
      stateMachine=stateMachine1:wpim::StateMachine {
        states=state1:wpim::State {
          presentation=presentation1:wpim::Presentation {}},
        stateMachine=stateMachine1{application=application1}
      };
    };
  };

  enforce domain aPSM siteMap2:apsm::SiteMap {
    name=applicationName,
```

```

navigationNodes=node2:apsm::NavigationNode {},
application=application2:apsm::Application {
  name=applicationName
};
where {presentation_node(state1, node2);}
}

relation presentation_node {
  presentationName:String;linkName:String;

  checkonly domain wPIM state1:wpim::State {
    presentation=presentation1:wpim::Presentation {
      name=presentationName},
    outgoing=outgoing1:wpim::Transition {name=linkName}
  };

  enforce domain aPSM node2:apsm::NavigationNode {
    name=presentationName,
    outgoing=outgoing2:apsm::Navigation {
      name=linkName,source=node2}
  };
  where {transition_link(outgoing1, outgoing2);}
}

relation transition_link {
  checkonly domain wPIM transition1:wpim::Transition {
    target=state1_1:wpim::State {
      presentation=presentation1:wpim::Presentation {},
      outgoing=outgoing1:wpim::Transition {
        target=target1:wpim::State {}}}}
  };

  enforce domain aPSM link2:apsm::Navigation {
    target=target2:apsm::NavigationNode {}};
}

relation usecase_usecase {
  usecaseName : String;

  checkonly domain wPIM usecase1 : wpim::UseCase {
    name = usecaseName,
    stateMachine = stateMachine1:wpim::StateMachine {
      useCase=usecase1}
  };

  enforce domain aPSM usecase2 : apsm::UseCase {
    name = usecaseName,
    stateMachine = stateMachine2:apsm::StateMachine {
      useCase=usecase2},
    controller=controller2:apsm::Controller{
      name=usecaseName+'Controller'}
  };
  where {
    statemachine_statemachine(stateMachine1, stateMachine2);}
}

relation statemachine_statemachine {
  statemachineName : String; stateName:String;
  checkonly domain wPIM statemachine1 : wpim::StateMachine {
    name = statemachineName,states=state1:wpim::State {
      name=stateName,stateMachine=statemachine1}
  };
}

```

```

enforce domain aPSM statemachine2 : apsm::StateMachine {
  name = statemachineName, states=state2:apsm::State {
    name=stateName,stateMachine=statemachine2}
};
where {state_state(state1, state2);}
}

relation state_state {
targetName:String;
checkonly domain wPIM state1:wpim::State {
  outgoing=outgoing1:wpim::Transition {
    target=target1:wpim::State {name=targetName}}
};
enforce domain aPSM state2:apsm::State {
  outgoing=outgoing2:apsm::Transition {
    target=target2:apsm::State {
      name=targetName,incoming=outgoing2},source=state2}
};
where {state_state(target1, target2);
transition_transition(outgoing1, outgoing2);}
}

relation transition_transition {
sourceName:String;targetName:String;
checkonly domain wPIM transition1:wpim::Transition {
  source=source1:wpim::State {name=sourceName},
  target=target1:wpim::State {name=targetName}
};
enforce domain aPSM transition2:apsm::Transition {
  source=source2:apsm::State {name=sourceName},
  target=target2:apsm::State {name=targetName}
};
when {state_state(source1, source2);                state_state(target1,
target2);}
}

top relation presentationState_presentationState {
presentationName:String;

checkonly domain wPIM state1:wpim::State {
  presentation=presentation1:wpim::Presentation {
    name=presentationName,
    uiComponents=uiComponents1:wpim::UIComponent {}}
};

enforce domain aPSM state2:apsm::State {
  presentation=presentation2:apsm::Presentation {
    name=presentationName,
    uiComponents=uiComponents2:apsm::UIComponent {}}
};
where {operationTrigger_signalEvent(state1, state2);
selectState_populationState(state1, state2);}
}

relation operationTrigger_operationTrigger {
triggerName:String;dataName:String;applicationName:String;

checkonly domain wPIM presentation1:wpim::Presentation {
  uiComponents=operationTrigger1:wpim::OperationTrigger {
    name=triggerName,
  dataComposite=dataComposite1:wpim::DataComposite {

```



```

    name=dataName},
    uiComponents=uiComponents1:wpim::UIComponent {}
  },
  presentationState=presentationState1:wpim::State {
    name=stateName,
    stateMachine=stateMachine1:wpim::StateMachine {
      name=smName,useCase=useCase1:wpim::UseCase {
        name=ucName,
application=application1:wpim::Application{
  name=applicationName}}}}
  };

enforce domain aPSM presentation2:apsm::Presentation {
  uiComponents=operationTrigger2:apsm::OperationTrigger {
    name=triggerName,
    uiComponents=uiComponents2:apsm::UIComponent {}
  }, presentationState=state2:apsm::State {
    name=stateName,
    stateMachine=stateMachine2:apsm::StateMachine {
      name=smName,useCase=uc2:apsm::UseCase {
        name=ucName,
        application=application2:apsm::Application {
          name=applicationName,
          dataEntities=dataEntity2:apsm::DataEntity {
            name=dataName,application=application2}}}}}}};
where {
  deleteQuery_dataEntity(operationTrigger1, application2);
  createQuery_dataEntity(operationTrigger1, application2);}
}

relation deleteQuery_dataEntity {
  dataName:String;componentName:String;

checkonly domain operationTrigger1:wpim::OperationTrigger {
  dataComposite=dataComposite1:wpim::DataComposite {
    name=dataName},
    uiComponents=uiComponent1:wpim::UIComponent {
      name=componentName,dataComposite=dataComposite1},
    dataOperation=dataOperation1:wpim::DataOperation {
      queryType=wpim::QueryType::remove},
    uiComponents=uiComponent2:wpim::UIComponent {
      dataComposite=dataComposite2:wpim::DataComposite {}
    }
  };

enforce domain aPSM application2:apsm::Application {
  dataEntities=dataEntity21:apsm::DataEntity {
    name=dataName,attributes=attribute2:apsm::Attribute {}},
  dataEntities=dataEntity22:apsm::DataEntity {}
  };
  where {select_integer(uiComponent1, attribute2);}
}

relation select_integer {
  componentName:String;
  checkonly domain wPIM uiComponent1:wpim::Select {
    name=componentName};
  enforce domain aPSM attribute2:apsm::Attribute {
    name=getAttributeName(componentName).firstToLower(),
    dataType=apsm::DataType::integer
  };
}

```

```

relation operationTrigger_signalEvent {
  operationTriggerName:String;uiComponentName:String;

  checkonly domain wPIM state1:wpim::State {
    name=sourceName,
    presentation=presentation1:wpim::Presentation {
      uiComponents=operationTrigger1:wpim::OperationTrigger {
        name=operationTriggerName,
        uiComponents=uiComponent1:wpim::UIComponent {
          name=uiComponentName}}
    },
    outgoing=outgoing1:wpim::Transition {
      target=target1:wpim::State {name=targetName}},
      stateMachine=sml:wpim::StateMachine {name=smName,
        useCase=uc1:wpim::UseCase {name=ucName,
        application=appl:wpim::Application {name=appName}}}
    }
  };

  enforce domain aPSM state2:apsm::State {
    name=sourceName,
    stateMachine=sm2:apsm::StateMachine{
      name=smName,useCase=uc2:apsm::UseCase {name=ucName,
      application=app2:apsm::Application {name=appName}}
    },
    outgoing=outgoing2:apsm::Transition {
      signalEvent=signalEvent2:apsm::SignalEvent {
        name=operationTriggerName,
        parameters=parameter2:apsm::Parameter {
          name=uiComponentName}},
      target=target2:apsm::State {
        name=sourceName+'_'+targetName,
        stateMachine=sm2,
        outgoing=outgoing21:apsm::Transition {
          source=target2,target=target21:apsm::State {
            name=targetName,stateMachine=sm2,incoming=outgoing21}}}}
    }
  };

  relation operationTrigger_transitionCallEvent {
    dataName:String;
    operationName:String;
    applicationName:String;
    usecaseName:String;
    sourceName:String;
    targetName:String;

    checkonly domain operationTrigger1:wpim:OperationTrigger {
      name=operationName,

      dataComposite=dataComposite1:wpim::DataComposite {
        name=dataName},
      dataOperation=dataOperation1:wpim::DataOperation {},
      presentation=presentation1:wpim::Presentation {
        presentationState=presentationState1:wpim::State {
          name=sourceName,
          stateMachine=stateMachine1:wpim::StateMachine {
            useCase=useCase1:wpim::UseCase {name=usecaseName,
            application=application1:wpim::Application{
              name=applicationName}}},
            outgoing=outgoing1:wpim::Transition {
              target=target1:wpim::State {name=targetName}}}}
    }
  };

```

```

enforce domain aPSM transition2:apsm::Transition {
  stateMachine=stateMachine2:apsm::StateMachine {
    useCase=useCase2:apsm::UseCase {name=usecaseName,
    controller=controller2:apsm::Controller {
      operations=operation2:apsm::Operation {
        name=operationName.trim()}},
    application=application2:apsm::Application {
      name=applicationName,
      services=service2:apsm::Service {
        name=dataName+'Service',
      supplierDependencies=supplierDependency2:apsm::Dependency {
        client=controller2,supplier=service2},
      application=application2}}}},
  callEvent=callEvent2:apsm::CallEvent {operation=operation2}
};
}

relation operationTrigger_serviceOperation {
  dataCompositeName:String;

checkonly domain operationTrigger1:wpim::OperationTrigger {
  dataComposite=dataComposite1:wpim::DataComposite {
    name=dataCompositeName},
  dataOperation=dataAssociation1:wpim::DataOperation {}
};

enforce domain aPSM service2:apsm::Service {
  name=dataCompositeName+'Service',
  dataComposite=dataComposite2:apsm::DataComposite {
    name=dataCompositeName,service=service2,
  supplierDependencies=supplierDependency2:apsm::Dependency {
    client=service2,supplier=dataComposite2}}};
}

relation selectState_populationState {
  stateName:String;applicationName:String;ucName:String;

checkonly domain wPIM state1:wpim::State {
  name=stateName,
  presentation=presentation1:wpim::Presentation {
    uiComponents=operationTrigger1:wpim::OperationTrigger {
      uiComponents=select1:wpim::Select {}},
    stateMachine=stateMachine1:wpim::StateMachine {
      useCase=useCase1:wpim::UseCase {
        name=ucName,
      application=application1:wpim::Application{
        name=applicationName}}}
};

enforce domain aPSM state2:apsm::State {
  presentation=presentation2:apsm::Presentation {
    uiComponents=operationTrigger2:apsm::OperationTrigger {
      uiComponents=select2:apsm::Select {}},
    incoming=incoming2:apsm::Transition {
      source=state3:apsm::State {
        outgoing=incoming2,
        callEvent=callEvent2:apsm::CallEvent {
          },
        stateMachine=stateMachine2:apsm::StateMachine {
          useCase=useCase2:apsm::UseCase {
            name=ucName,
            application=application2:apsm::Application {

```

```

    name=applicationName},
    controller=controller2:apsm::Controller {
      operations=operation2:apsm::Operation {}}}},
    target=state2}
  };
}

top relation dataComposite_dataEntity {
  dataName1:String;dataName2:String;
  checkonly domain wPIM presentation1:wpim::Presentation {
    uiComponents=uiComponent1:wpim::UIComponent {
      dataComposite=dataCompositel:wpim::DataComposite {
        name=dataName1,
        selectionBase=dataCompositel1:wpim::DataComposite {
          name=dataName2}}}
  };
  enforce domain aPSM application2:apsm::Application {
    dataEntities=dataEntity2:apsm::DataEntity {
      name=dataName1,
      attributes=attribute2:apsm::Attribute {}},
    dataEntities=dataEntity21:apsm::DataEntity {
      name=dataName2}
  };
}

query getDataOperationName():String {
  if (dataOperation=wpim::QueryType::remove) then 'remove'
  else if (dataOperation=QueryType::create) then 'create'
  else if (dataOperation=QueryType::update) then 'update'
  else if (dataOperation=QueryType::select) then 'select'
  else 'selectAll' endif endif endif endif }

query getAttributeName(componentName:String):String {
  componentName.split(' ')->iterate(s:String;acc:String='')|
  acc.concat(s.firstToUpper())}}

```