Towards A Semi-Automated Model-Driven Method for the Generation of Web-based Applications from Use Cases

Ali Fatolahi[1], Stéphane S. Somé[1], and Timothy C. Lethbridge[1]

School of Information Technology and Engineering, University of Ottawa
{afato092, ssome, tcl}@site.uottawa.ca

Abstract. This paper presents a semi-automated method for the generation of web-based applications from high-level requirements expressed as use cases in accordance with model-driven architecture (MDA). MDA is a relatively new paradigm, which aims at providing a standard baseline for model-driven development. The ultimate goal of MDA is to (semi)automate the process of software development from requirements to code using an interoperable set of standards. Being very popular, use case modeling is a perfect choice for capturing requirements at the beginning of an MDA-based process. We consider the use case model as a baseline to generate other models including a state machine and a user interface model, which are eventually transformed into a platform-specific model used for code generation.

Keywords: MDA, Use Case, PIM, PSM, Web Application, Transformation

1 Introduction

MDA [1] is the OMG's [2] solution to increase model reusability and design-time interoperability. A very important feature of MDA is a facility to transform models. Not only is it easier to build automatic model mappings in the MDA context, but MDA could also be beneficial when the model transformation is done manually. MDA provides a collection of popular standards beneath a common philosophy to alleviate the process of quality software design and implementation. There has been a growing interest in MDA within the software community in the recent years [3].

The MDA process starts with capturing requirements at a computation-independent layer [3]. In our approach, use case descriptions are used for requirements capture. Use cases are a popular technique for systems analysis and design that are mainly expressed using informal or semi-formal textual descriptions.

However, since writing textual descriptions is not as formal a task as drawing UML [4] models and writing programming code, different guidelines have been proposed to ease the process of writing use case descriptions and benefiting from these documents (e.g. [5] and [6]). The fact that several approaches have examined semi-automatic use-case based tools/techniques (e.g. [7], [8] and [9]) evidences that use cases as a technique for analysis would be more useful, if we find some ways to connect them to lower-level design models or even to code.

In this paper, we extend our previous work [34] to present a method for the semi-automated generation of design models related to web-based applications from requirements. Requirements are expressed as use case descriptions along with a domain model supporting the use cases. The whole model is used to produce a state machine. A default user interface model created based on the state machine is refined by the developer to form the desired user interface of the application. Based on these models, the method generates a platform-specific model, which is used to generate the code. The developer is our preferred name for the user of our method.

In order to assess the feasibility of the approach we have implemented the method using UCEd [10] and AndroMDA [11]. UCEd is used for use case modeling and AndroMDA for code generation. In addition, we have developed an application to generate the platform-specific model that bridges the output of UCEd to the input of AndroMDA. However, the method and supporting tools and techniques are supposed to be extensible to higher-level requirements and adaptable with other tools. Because of our previous experience and familiarity with UCEd and AndroMDA as elaborated in [7] and [34], we preferred to use those tools for the current stage so that we could focus on the core ideas rather than learning new tools.

The rest of this paper is organized as follows. In Section 2, technical background of this research are explained. Section 3 carries the elaboration of our method and the applied tools and techniques along with a case study. Section 4 addresses related research and practice in past and how they are related to this research. In Section 5, we, briefly provide a conclusion, discuss some research issues and present our plan for future work.

2 Background

MDA is an effort by OMG, in order to standardize model driven software development [15]. It could be seen as a framework, composed of four different layers of modeling. The most top layer is the layer of Computation-Independent Models (CIM). This layer represents models, which are valid in spite of the computational options. Then we have the layer of Platform-Independent Models (PIM). PIM acts as a standpoint of systems/software design and architecture. However, it does not contain any information about specific platforms. The third layer, Platform-Specific Models (PSM) deals with the technological details of platforms. Here, logical design models are expressed in terms of certain platforms.

In our research, we also use the method presented by Somé in [7], which is done with the help of the UCEd. This method elaborates the necessities to support use-case based requirements engineering. This support is given throughout domain objects, operation (pre and post)conditions and semi-natural language use case steps; for each of which, UCEd provides some automatic and/or semi-automatic means. The output is a state machine that belongs to the category of platform-independent models, since it sketches an overview of how the system works without any design-related details.

In order to work with UCEd, one needs to first enter use case descriptions. Having this description validated, she could go through a wizard in which UCEd provides her with a series of different choices for domain objects. The result is a validated domain model. This domain may be optionally supplied with operations' conditions that are used to build some form of operation contracts [27]. State machine could be generated thereafter.

3 The Method

The solution we provide in here is a method, which is both model driven and requirements based. The input is provided through use cases and the output is the executable code generated in accordance with MDA. Different steps of the method are either automatic or semi-automatic. The whole process is actually a collection of mappings in accordance with XMI format necessities, MOF-based metamodels and MDA transformation rules.

Figure 1 is a decomposition of the method into three main steps. In Figure 1, steps A2 and A4 could be understood as the UCEd and AndroMDA processes. Step A3 is the core of our method, which is described using more details in Figure 2.

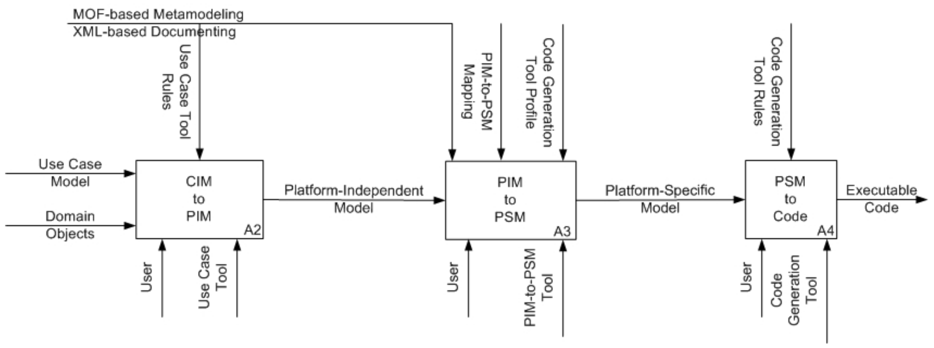

Fig. 1. Three main steps to generate code from use cases

As Figure 2 shows, the main task of this process is to transform a PIM to a PSM. This process includes three main steps. At first, a default UI model is created according to the state machine found in the PIM. The developer is then asked to refine this model to build her desired UI model. Finally the UI model, along with other parts of the PIM, is used to generate a PSM. The manual work happens in the first and last step, where the developer has to interact with the tools to generate the state machines as well as to refine the design-specific models required for code generation. The middle step is the main contribution of this paper.

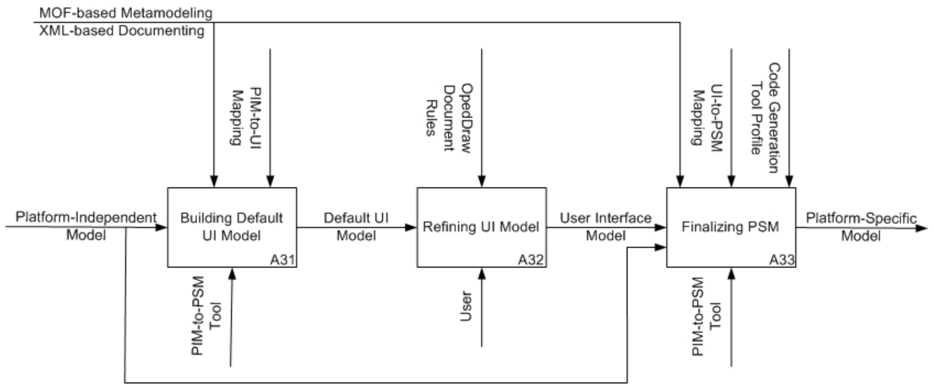

Fig. 2. Transforming a PIM to a PSM

Having the whole method depicted, we are now able to go through the details of the method, mappings and the techniques used to steer the process. This is done using a working example, named Election Management System(*EleManSys*) taken from [35] throughout this paper. The EleManSys is a system used for managing elections and their related polls. The EleManSys is composed of several use cases required for performing operations by election officers, candidates, voters and journalists who may use the reports for analysis purposes.

3.1 First Step: CIM to PIM

Use case descriptions and default domain objects are considered as the CIM in our method. The objective of this step is to transform the CIM to the PIM. The PIM includes the state machine, the user interface model and the refined domain model. By refined domain model, we mean a domain model that is enriched with the operations of objects, operation conditions and fundamental attributes of objects.

The CIM could be already generated by another tool or alternatively built up from scratch using UCEd. The generated UCEd model is actually an XML file containing the use case model and domain objects. However, the state machine model should be manually transferred into a text file for further use. Because of the implementation limitations, the user interface model generation, which is theoretically a part of this step, is done in the next step.

Table 1 lists a few use cases included in the CIM of EleManSys. This CIM is transformed to the PIM afterward. Figure 3, shows a part of the PIM, which is a state machine that models the flow of the *Open Poll* use case. Since this use case includes two other use cases, the state machine covers the included use case as well.

Table 1. Use Cases of EleManSys

Name	Description
Open Poll	Pre-Condition: EleManSys is Up 1. Election Manager selects Open Poll 2. EleManSys shows the list of polls 3. Election Manager selects a poll 4. EleManSys opens poll 5. Include View Poll Results Alternatives: 2.a. if polls list is empty 2.a.1. Include Add Poll 2.a.2. goto 4 Postcondition: poll is open
Add Poll	Pre-Condition: EleManSys is up 1. Election Manager selects Add Poll 2. EleManSys opens add poll view 3. Election Manager enters Poll 4. Election Manager confirms Add Poll 5. EleManSys adds Poll to elections DB 6. EleManSys shows add poll success message Post-Condition: poll is added
View Poll Results	Pre-Condition: Poll is Open 1. Election Manager selects View Poll Results 2. EleManSys shows poll results 3. Repeat every 2 min 3.1. EleManSys updates poll results Post-Condition: poll results is updated
Vote	Pre-Condition: Poll is Open 1. EleManSys prepares default vote 2. Voter selects candidate and seat 3. Voter selects Add Vote 4. EleManSys adds vote 5. EleManSys shows success message Post-Condition: success message is shown

EleManSys has several other use cases that are not covered in this paper. These use cases include:

– A use case for the reporters to keep track of incumbents for a seat, so that they can report the incumbent's possible loss or win.
– A use case to define eligible voters. Every voter can only vote in certain polls for specific seats.

– A use case to close a poll, which includes the *View Poll Results* use case.
– Other administration use cases to create, delete or update the information regarding Elections, Polls, Seats, Candidates and Voters

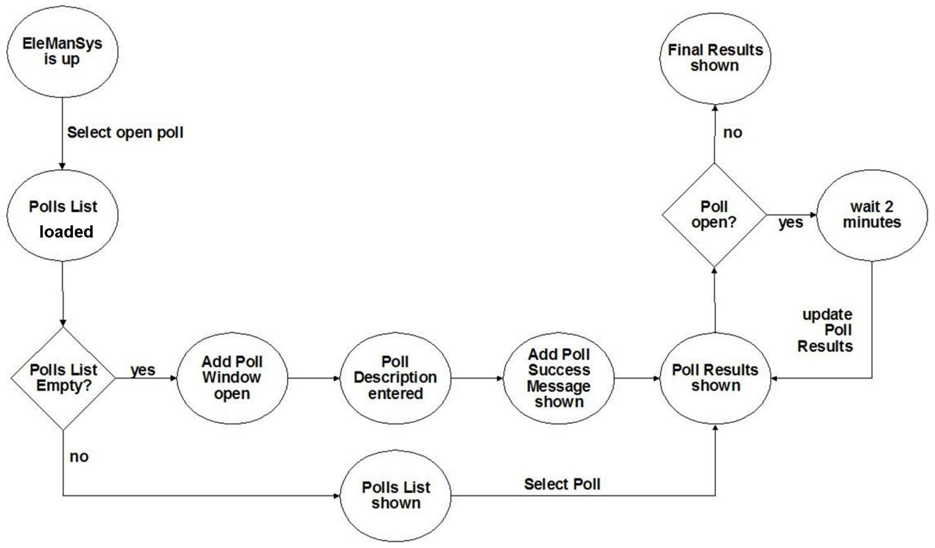

Fig. 3. The state machine generated by UCEd for the use case *Open Poll*

The state machine leads to the information regarding the domain objects and their operations, the flow of events and the conditions that apply to every phase. In order to generate the state machine using UCEd, one can take one of the two following approaches: One is simply the basic state machine generated from the use case flow. The more formal way; however, is to generate the state machine based in the operation conditions. UCEd performs this type of state-machine generation in the following way:

– During the domain extraction wizard, domain operations are extracted and assigned to domain entities.
– The developer defines operation conditions so that different operations' conditions match their preceding and following operations.
– The state machine is generated starting from the first use case step following each operation's condition that matches the previous one or by the usual use case flow of events.

3.2 Second Step: PIM to PSM

A PIM-to-PSM is a mapping from a platform-independent model of domain objects, state machines and user interface models to a platform-specific model. The true mapping should be done according to a selected profile. However, due to implementation constraints, we have to pick up some profiles already created to be used with a specific code generation framework, AndroMDA. The chosen profile contains various definitions about Java as the programming language, AndroMDA as code generation framework, ArgoUML as the modeling tool, Struts [25] as the user interface framework and MySQL [26] as the database server.

The resultant application follows an MVC-based architecture with data access mechanisms - so called *services* - at the lowest level. The flow of data between different layers is facilitated using *value objects*. The behavior is controlled using *controller* classes and operations. Related definitions are as follows:

- A *State Machine Context* is a controller class that is responsible for handling and forwarding the operations regarding events occurring within the state machine.
- A *controller* class is the class responsible for controlling general activities within a use case. Controller can act as an entry point to dispatch messages and operation requests to the right target. This is in accordance with a design pattern with the same name. For more information regarding the controller pattern, see [27].
- An *action event* is an event to be called when submitting a form from within a web page. Action events usually include parameters which are the input fields on the forms.
- A *deferrable event* is an event calling a controller operation. Deferrable events are used to assign states with operations.
- A *page parameter* is any output that is either shown or used for other output fields on the web page.
- A *value object* is an object to carry the required 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 the event that is usually carried by a transition incoming to a front-end state, carrying the output fields to be shown.
- A *call event* is an event located on a transition outgoing from a front-end state, carrying the input fields to be submitted along with the controller operation to be called for performing the required action. For more information, one may refer to [28].

The approach is not fully automatic; we need to interact with the developer to obtain the appropriate value of required user parameters. In order to do this, we first create a default UI model according to the provided PIM state machine. This default UI model will be the base model to generate the PSM thereafter.

The method is adjusted to work with the OpenDocument format [29]. OpenDocument is an XML-based format to define documents containing text or graphics. This means that the user interface model should be created in accordance with this format. Currently, we use OpenDraw [30] to draw the UI model. The developer is required to create the UI model following some rules:

- Every non-empty slide is considered a presentation state (e.g. a web page)
- Drawing items could be grouped to represent a group of related outputs on a page or more importantly an input action and its related items (e.g. a submit button)
- A frame is a symbol of an action
- A triangle represents a dropdown input
- A rectangle represents a plane text input
- A cloud could be grouped with any input item to identify its data type
- A ring can be grouped with an action denoting the called operation associated with the action
- A cylinder could be grouped with any input item declaring its data source containing the name of table and/or column
- If a group of outputs was combined with a cross, this would mean a table view output

States are recognized as either presentation (front-end) or logic states. For each presentation state, s_i

- s_{i-1} is the state preceding s_i
- s_{i+1} is the state following s_i
- $t_{i-1,i}$ is the transition from s_{i-1} to s_i
- $t_{i,i-1}$ is the transition from s_i to s_{i-1}
- $t_{i,i+1}$ is the transition from s_i to s_{i+1}
- $t_{i+1,i}$ is the transition from s_{i+1} to s_i

Neither s_{i-1} nor s_{i+1} could be a presentation state. Currently we accept no multiple transitions out of a state, which means there would be just one form per web page.

Figure 4 shows the UI model of the state, *Polls List Shown* of EleManSys (see Figure 3), where the User is asked to select a poll to open. The default user interface model includes an empty slide for each step of the state machine in Figure 3. The developer has then generated the user interface model for this state machine assigning the model in Figure 4 to the step, *Polls List Shown*. According to this figure, EleManSys shows a webpage containing an action form submitting a poll selected from the list of polls coming from database table *polls* to the operation, *openPoll*.

Suppose A as the set of actions submitted from s$_i$. Each action is defined as the tuple $\{a,\ F_i,\ operation\}$. F_i is the set of input fields and is defined as $\{\{F,r\}:\{f,r\}^+\}$ in which

- F is a set of fields
- f is a single field
- r is the value object providing the input
- *operation* is the name of the operation this action calls.

Also, assume O as the set of outputs shown by s_i. Each output is simply a set of output fields, F_o.

Back to the EleManSys example, we can see that for s2_0, A includes one action that is defined as $\{$*Open Poll,* $\{\{$*poll, POLLS*$\}$*, openPoll*$\}$, where

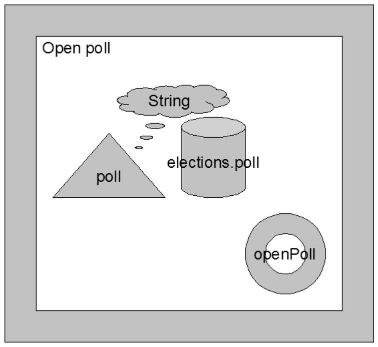

Fig. 4. *Polls List Shown* state: UI Model

- 'Open Poll' is the action
- The set of input fields (F_i), includes
  - *poll* coming from database table *POLLS*
- And *openPoll*, the controller operation to be called

There are some mapping rules that apply regardless of the UI model:

- A main use case is defined and stereotyped as *FrontEndApplication* to indicate the entry point of application. The execution of other use cases will originate from this use case. This main use case will be supported by a state machine, which has:
  - a *FrontEndView* state
  - an outgoing transition for each use case; this will carry a signal event named after the action event of the corresponding use case and will end to a final state named after the corresponding use case.
  - for use cases that have a pre-condition, a junction will be added to check if the pre-condition is met? If not the flow simply returns to the main state. This junction is preceded by another state. The transition from the state to the junction will have a call event that runs a controller operation to check the pre-condition.
- For every use case that has a state machine including at least one presentation states, a use case is created with the stereotypes, *FrontEndUseCase*
- There must be one controller class per use case
- There must be a service class per use case
- Controller classes must be dependent on their objects' service classes

Figure 5 shows the main state machine of EleManSys imported and refined using ArgoUML.

Other mapping rules are as follows

- Every presentation state becomes a state stereotyped as *FrontEndView*
- For each member of A
  - a becomes an action event on $t_{i,i-1}$ or $t_{i,i+1}$ whichever exists

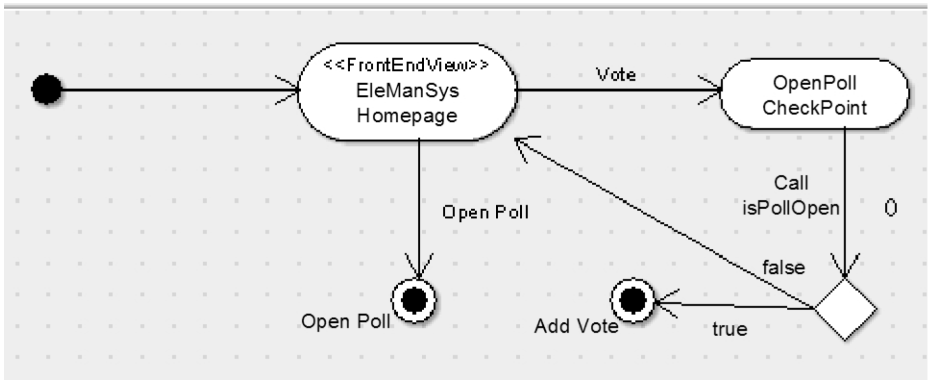

Fig. 5. The Main State Machine of EleManSys

  - a becomes a deferrable event on $_{si+1}$ calling *operation*
  - *operation* becomes an operation of the controller class
  - F_i becomes the set of input parameters on both a and *operation*
- For every database table referred by members of F_i, a domain object and a value object are created
  - There would be a dependency from every entity domain object to the relevant value object
  - Add an operation to the service class to retrieve the data from database
  - Make a dependency from the service class
- For each member of O, F_o becomes the set of parameters on a signal event belonging to $t_{i-1,i}$ or $t_{i+1,i}$ whichever exists

The mapping rules have been also expressed using Query-View-Transformation (QVT) [22] language to make sure the method can integrate with other MDA-based models and methods.

Figure 6 shows a simplified version of the state machine generated for the use case, *Open Poll*, which includes the *View Results* and *Add Poll* use cases as well. The state machine was generated by the method and then imported and refined using ArgoUML. Some details are shown for more clarification. For example, Figure 6 shows that the transition after the first FrontEndView has a signal event carrying the parameter, *poll*, which means the there would be an entry named, *poll* on the corresponding web-page. Figure 7 depicts another part of the PSM, which describes the service class model of the *Open Poll* use case.

3.3 Third Step: PSM to Code

This last step is done by the code generation tool. Currently, we generate a PSM that could be edited by ArgoUML and read by AndroMDA. Developers are given the chance to refine the generated model or simply generate and launch the executable code.

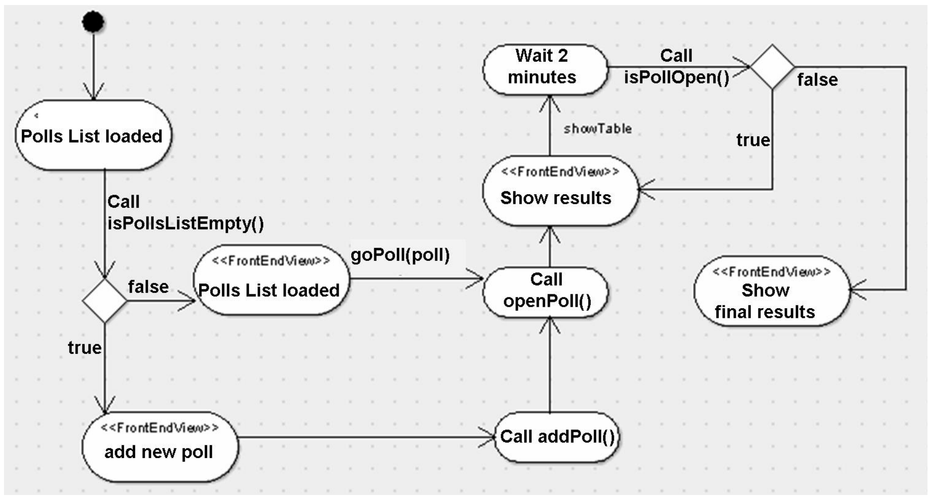

Fig. 6. Open Poll State Machine

4 Related Work

A model-driven approach for the semi-automated generation of web-based applications using UWE [36] is introduced by Kraus et al [21]. This method goes through requirements analysis, conceptual design, navigation design and presentation design in order to build the application. Automated transformations can 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 approach is an adequate method to generate web-based applications that provides detailed mechanisms to define the application 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, Karus et al's effort is dedicated to preparing a framework based on which the 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 for others to build methods on top of ours; 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.

Another related work is presented by Wu et al [22]. This work describes a method to generate a user interface code following MDA transformation 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 a robustness analysis [23] to JSP pages [24]. In order to do this, the authors provide a framework that starts with use case modeling and activity

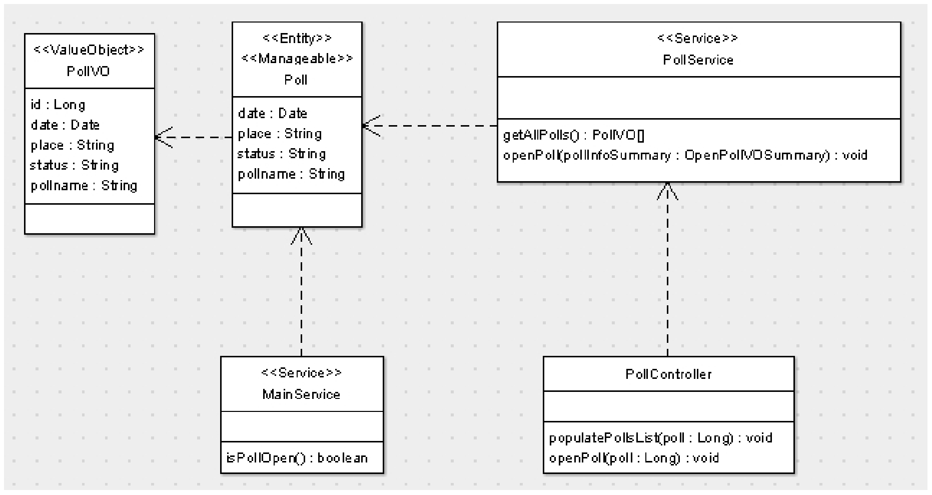

Fig. 7. Platform-Specific classes related to the *Open Poll* use case

diagram. Then they perform a robustness analysis to categorize the participating objects. Finally, JSP pages are built according to the transformation rules and UML models.

Unlike Wu et al's work, our study 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 theoretically platform-independent. Finally, we have developed not only a method but also a practice to show the feasibility of the method, which is not presented by Wu et al.

Brambilla et al. [17] introduce an approach to web development using WebRatio [18] as a tool and WebML [19] 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
– Integrating Web services

The resultant PSM will be automatically mapped to the executable code.

Brambilla et al's approach is focused on the automation of the PSM-to-Code transformation while providing a detailed mechanism for developing the PSM. Our approach is instead centered on the PIM-to-PSM transformation and leaves the code generation to other tools. WebRatio is a good alternative to the AndroMDA-based platform we use at the moment.

In [31], authors suggest a model-drievn method for the generation of web-based applications using OOHDM [20], which is a data centric and object oriented model for hypertext modeling. 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 of this approach to ours is the lack of automation especially in terms of data access and UI modeling.

Another approach based on OOHDM is published in [32]. This approach results in two different PSMs: conceptual and navigational, which has the advantage of the choice of different independent technologies as for presentation and logic. However, the approach only covers the navigational transformations. Navigation is separated into that for fixed and dynamic pages. 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 this approach shows to our method is the fact that the UI model have not been considered as a requirement. Generally speaking, authors are not concerned with requirements in their method.

We may also mention [33], which is a conceptual framework for MDA-based software development environments. The approach presented in their book is neither rendered in practice nor formally; however since an ultimate goal of our method could be the generation of an MDA-based environment, their work provides a useful point of reference.

Some related works present similar features to ours but our approach remains unique especially in terms of automating the PIM-to-PSM transformations and considering UI models as a requirement.

5 Conclusion

In this paper we presented the results of the research and development of a method to address the issue of the generation of web-based applications from requirements. This is done using several transformations over use cases, user interface models, state machines, design models and code. The output of our method is a PSM that could be used to generate executable application.

Although the intention is to implement a general-purpose method but our conclusion shows that the method shows better compliance with data-centric web applications that contain several database-related activities. As a result, our future work is mostly devoted to automating the process of generating data operations and mechanisms in short term. In the long term we will look at other possibilities.

A Java application has been implemented to run the method using real examples. Besides the *EleManSys* example, our tool has so far been tried on several other examples taken from actual case studies including the ones found in [11]. Other examples have been either planned or queued to test the method.

As future work, we also intend to cover more complicated cases. We are especially interested in mixed problem classes, applications with database transactions and multiple use cases and state machines within the same application.

We also intend to evaluate the effectiveness of the method by having groups of developers evaluate it in practice.

One critical feature to add to the method is the ability of the method to identify manageable entities. A manageable entity is required whenever the developer needs to create, delete or update an entity; this is done by stereotyping the entity as *Manageable*. Regular *Entity* stereotype can only support the database query operations. An example is the Poll entity in Figure 7. Currently, we are working on adding a tagged value to use case steps for this purpose.

The UI model used in this paper needs to be formally defined. To this end, we will choose an existing abstract UI model that could be used in an MDA-based environment to define the UI model. One example is introduced by Koch [12].

Finally, in order to generalize the approach it is critical to employ an abstract model that describes web-based applications. There are many models for this purpose and the selection of a suitable candidate needs an in-depth study. So far, we have found the model provided by Baresi et al [13], called W2000, a good candidate as it adapts with both MDA and MVC.

References

1. Model-Driven Architecture, www.omg.org/mda
2. Object Management Group, www.omg.org
3. Meservy, T.O., Fenstermacher, K.D.: Transforming software development: an MDA road map. IEEE Computer, 38 (9), 52–58 (2005)
4. Object Management Group UML, www.uml.org
5. Cockburn, A.: Writing Effective Use Cases. Addison-Wesley, Harlow (2001)
6. Chen, Y., Song, I.Y.: Guidelines for Developing Quality Use Case Descriptions.In: IRMA International Conference, pp 564–567 (2007)
7. Somé, S. S.: Supporting use case based requirements engineering. Information and Software Technology, 48 (1), pp. 43–58 (2006)
8. Behrens, H.: Requirements Analysis Using Statecharts and Generated Scenarios. In: Doctoral Symposium at IEEE Joint Conference on Requirements Engineering (2002)
9. Sutcliffe, A. G. Maiden, N. A. M., Minocha, S. Manuel, D.: Supporting Scenario-Based Requirements Engineering. IEEE Transactions on Software Engineering, 24 (12), pp. 1072–1088 (1998)
10. Use Case Editor, www.site.uottawa.ca/s̃some/Use_Case_Editor_UCEd.html
11. AndroMDA, www.andromda.org
12. Koch, N. Software Engineering for Adaptive Hypermedia Systems: Reference Model, Modeling Techniques and Development Process PhD. Thesis, FAST Reihe Software technik, UNI-DRUCK Verlag, December 2000.
13. Baresi, L. Colazzo, S. Mainetti, L. and Morasca, S. W2000: A Modeling Notation for Complex Web Applications. In E. Mendes and N. Mosley (eds.) Web Engineering: Theory and Practice of Metrics and Measurement for Web Development. Springer, ISBN: 3-540-28196-7, 2006.
14. UML, Unified Modeling Language: Superstructure version 2.1.1 (non-change bar) formal/2007-02-05 (Available from: www.omg.org/cgi-bin/doc?formal/07-02-05)
15. Stahl, T. and Volter, M, Bettin, J. Haase, A. and Helsen, S.: Model-driven software development : technology, engineering, management /translated by Bettina von Stockfleth. John Wiley, Chichester, England ; Hoboken, NJ (2006)

16. QVT 1.0, www.omg.org/spec/QVT/1.0, May 2006.
17. Brambilla, M. Comai, S. Fraternali, P. and Matera, M. Designing Web Applications with WebML and WebRatio. In book: "Web Engineering: Modelling and Implementing Web Applications." Gustavo Rossi, Oscar Pastor, Daniel Schwabe and Luis Olsina. 2007
18. WebRatio, www.webratio.com
19. WebML, www.webml.org
20. The Object-Oriented Hypermedia Design Model (OOHDM), www.telemidia.puc-rio.br/oohdm/oohdm.html
21. Kraus, A. Knapp, A. and Koch, N. Model-Driven Generation of Web Applications in UWE. In Proc. MDWE 2007 - 3rd International Workshop on Model-Driven Web Engineering, CEUR-WS, Vol 261, July 2007
22. Wu, J. H., Shin, S. S., Chien, J. L., Chao, W. S. and Hsieh, M. C.L An Extended MDA Method for User Interface Modeling and Transformation. In: The 15th European Conference on Information Systems. pp 1632–1641 (2007)
23. Rosenberg, D. and Stephens, M.: Use case driven object modeling with UML : theory and practice, Apress Publishers (2007)
24. Java Server Pages Technology, java.sun.com/products/jsp
25. Struts, struts.apache.org
26. MySQL AB, dev.mysql.com
27. Larman, C.: Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Prentice Hall PTR (2005)
28. AndroMDA BPM4Struts, galaxy.andromda.org/docs/andromda-bpm4struts-cartridge/index.html
29. OASIS, www.oasis-open.org
30. Draw, /www.openoffice.org/product/draw.html
31. Rossi, and G. Schwabe, D. Model-Based Web Application Development. In Mendes E. and Mosley N. (eds.) Web Engineering: Theory and Practice of Metrics and Measurement for Web Development. Springer, ISBN: 3-540-28196-7, 2006.
32. Schmid, H. A. and Donnerhak, O.: The PIM to Servlet-Based PSM Transformation with OOHDMDA, Workshop on Model-driven Web Engineering (MDWE 2005) July 26, 2005
33. Pastor O. and Molina J. C., Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling, Springer (2007)
34. Fatolahi A., Somé S. S. and Lethbridge T. C. A Model-Driven Approach for the Semi-Automated Generation of Web-based Applications from Requirements. Proceedings of 2008 International Conference on Software Engineering and Knowledge Engineering. pp 619-624.
35. Lethbridge T. C., Laganire R., Object-oriented software engineering : practical software development using UML and Java, London : McGraw-Hill, c2001
36. UWE UML-based Web Engineering, www.pst.informatik.uni-muenchen.de/projekte/uwe/