

An Abstract Meta-Model for Model Driven Development of Web Applications Targeting Multiple Platforms



Ali Fatolahi

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements
for the PhD degree in Computer Science

Ottawa-Carleton Institute for Computer Science
School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

© Ali Fatolahi, Ottawa, Canada, 2012

Abstract. In this thesis, we present an abstract meta-model for model driven development of web applications targeting multiple platforms. We review the existing technologies and the related work in order to obtain a list of requirements for such an abstract model. The abstract model is built by extending an existing UML-based model for web applications. We demonstrate that it is possible to map this abstract model to more than one specific development platform by providing transformations for these mappings. We also lay out the general outline of a model-driven process based on the proposed abstract model. The abstract model and the model-driven process are supported by a set of tools, case studies and a visual modeling notation.

Model-driven techniques have been used in the area of web development to a great extent. Most of the existing approaches are tuned toward specific platforms or develop only certain parts of web applications. These approaches generally use meta-models adapted to their targeted platforms. In order to flexibly target multiple platforms, the level of abstraction of the meta-model must be raised. Such a meta-model must allow the description of relevant features of web applications independently from the specificities of specific platforms. Additionally, transformations mapping from abstract to specific web descriptions must be expressible in a flexible way. In this thesis, we propose such an abstract meta-model. Mappings that transform abstract models to specific platforms are also presented.

Different benefits can be foreseen from this 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 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. 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 this approach is its use of a specification of the data mapping as part of its high-level input. More importantly, the common features required to process data and communicate data objects between different layers and components are targeted.

Acknowledgements. This dissertation would not have been possible without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study. First of all, I wish to express my gratitude to my supervisors Stéphane S. Somé and Timothy C. Lethbridge, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject.

Further, I would like to offer my regards and blessings to the members of the committees of my comprehensive examination, thesis proposal defense as well as the final thesis defense for their very helpful comments and recommendations. I also extend my best wishes to all my friends, who supported me in any respect during the completion of the project especially my dear friends Saeed Ahmadi Behnam for peer-reviewing the thesis and Shahram Jalalinia for providing me the access to one of the major case studies.

The last but not the least, I am heartily thankful to my beloved wife for her understanding and endless love, through the duration of my studies.

Table of Contents

Chapter 1. Introduction.....	1
1.1 Motivation.....	1
1.2 Problem Statement.....	4
1.3 Hypothesis.....	4
1.4 Solution Layout.....	4
1.5 Research Methodology.....	7
1.6 Contributions.....	8
1.7 Outline.....	12
Chapter 2. Background.....	13
2.1 Separation of Concerns.....	13
2.1.1 Model-View Controller (MVC).....	14
2.2 Model Driven Development (MDD).....	16
2.2.1 Model-Driven Architecture (MDA).....	17
2.2.2 PIM and PSM.....	18
2.2.3 Platform.....	21
2.3 Use Cases.....	23
2.4 State Machines.....	25
2.5 Information Systems.....	27
2.6 Transformations.....	28
2.6.1 Transformation Languages.....	29
2.6.2 Visual Transformation Languages.....	32
2.6.3 Types of Transformations.....	33
2.6.4 Transformation Implementation Tools.....	34
2.7 User Interface Modeling.....	34
2.8 Web Engineering.....	36
2.9 Summary.....	38
Chapter 3. Literature Review.....	39
3.1 Requirements of an Abstract Web Model.....	39
3.2 General MDWE-Related Approaches.....	43
3.2.1 Related Approaches.....	43
3.2.2 Other Approaches.....	44
3.3 Related Approaches.....	47
3.3.1 Unified Web Engineering (UWE).....	47

3.3.2	Object Oriented Hypermedia Design Method (OOHDM).....	49
3.3.3	Wu et al.'s approach	50
3.3.4	Web Software Architecture (WebSA)	51
3.3.5	Ubiquitous Web Application (UWA)	52
3.3.6	Muller et al's approach	53
3.4	The Web Modeling Language (WebML).....	54
3.5	The Botterweck Model.....	57
3.6	Comparative Study of the Existing Approaches	59
3.7	Summary	63
Chapter 4.	The APSM	64
4.1	Basic Terminology	64
4.2	A High-Level View	65
4.3	Web Application Use Case Model	66
4.4	Web Application Back-End	70
4.5	An Interactive Model	72
4.6	Site Map	73
4.7	Web Application Security	73
4.8	A Sample Authentication Model.....	74
4.9	A Data-Driven Example.....	78
4.10	A Multi-Layered Model	78
4.11	Extending the Botterweck Model.....	79
4.12	Summary	83
Chapter 5.	The PIM	84
5.1	A Visual Input Language	84
5.1.1	Components	84
5.1.2	Information Elements.....	85
5.1.3	Logic	86
5.1.4	UML.....	87
5.1.5	Semantics of a <i>pim</i>	87
5.2	PIM as the Meta-Model of the Visual Language	89
5.3	Examples	92
5.3.1	Example 1: Adding Comments to a Content-Oriented UI Element	93
5.3.2	Example 2: Sequencing State Machines	94
5.4	Summary	97

Chapter 6.	Mapping the pim to the apsm.....	98
6.1	PIM-to-APSM Mapping Rules	98
6.1.1	Mapping Data Entities	103
6.1.2	Mapping Signal Events	104
6.1.3	Mapping Call Events.....	104
6.1.4	Mapping Controller Operations	105
6.1.5	Mapping Services.....	106
6.1.6	Mapping Domain Objects	107
6.2	PIM-to-APSM QVT Relations.....	108
6.3	A Sample Relation	111
6.4	Summary	116
Chapter 7.	Transforming to Specific Platforms.....	118
7.1	APSM-to-AndroMDA.....	118
7.1.1	APSM-to-AndroMDA QVT Relations	120
7.1.2	A Sample APSM-to-AndroMDA Relation.....	122
7.1.3	A Sample AndroMDA Model.....	125
7.2	APSM-to-WebRatio	126
7.2.1	APSM-to-WebRatio QVT Relations	128
7.2.2	A Sample WebRatio Model	130
7.3	APSM-to-GWT	130
7.4	APSM-to-Net	135
7.5	Summary	137
Chapter 8.	Implementation: A Visual Overview	139
8.1	The Implementation Suite	139
8.2	The Election Management System.....	140
8.3	Visual PIMs.....	141
8.4	Generating PIM EMF	142
8.5	Mapping to APSM	147
8.6	Mapping to GWT	151
8.7	Summary	157
Chapter 9.	Evaluation	158
9.1	Abstract Web Criteria.....	158
9.2	Mapping to Multiple Platforms	160
9.3	Separation of Concerns	162

9.4	Handling Requirements.....	164
9.5	Methodological Evaluation	165
9.5.1	A Sample Use Case.....	170
	Operation Contracts.....	170
9.6	Case Studies	173
9.7	Summary	179
Chapter 10.	Conclusion.....	180
10.1	Remaining Issues.....	180
10.2	Future Work	181
References		184
Appendix A.	Class Descriptions	202
Appendix B.	Dilmaj Case Study.....	237
Appendix C.	AMS Case Study	276
Appendix D.	EMS Case Study.....	340
Appendix E.	Glossary.....	356

Chapter 1. INTRODUCTION

Several model-driven development (MDD) techniques for web applications exist; these techniques use meta-models for defining transformations and designing models. For the most part, these models are either platform-specific or do not support mappings to multiple web platforms. In this thesis, we propose an abstract web model that can be mapped to multiple platforms. In this chapter, we establish the need for such models. We state the problem and formally state a hypothesis. Afterwards, we sketch the layout of a solution and we present the list of contributions of this thesis.

1.1 Motivation

The gigantic size of the web and its growth rate indicate the importance of web applications and services. According to Netcraft [NetCraft2011], nearly 300 million web servers across the world have responded to their Web Server Survey in March 2011, among which more than 110 million servers are recognized as active web servers. In addition, these numbers are increasing; in March 2011, for example, there has been an increase of 13 million new hostnames compared to February 2011.

Web servers are typically used to host web sites and provide web services. These web-based technologies act as facilitators that add to the value of the web by connecting people, enabling low cost business and sharing information. Static web sites that are composed of a few HTML pages with minor user interaction are becoming less prevalent. A great majority of the current web sites are dynamic and support web applications. Web services are also mostly used by other web applications that call them. Developers contribute to this ever-increasing network using a variety of technologies under different configurations.

While the number of new web applications is increasing, existing web applications also must be constantly updated to conform to newer technologies or changes to the existing technologies. Even the existing similar technologies are being used in a variety of different configurations in a manner that impedes portability. For example, Google Web Toolkit (GWT) [GWT2010] and WebRatio [WebRat2008] are both Java-based but there is no way to reuse a

web page created using one technology in the other¹. This illustrates the difficulty of switching from one technological platform to another. Developers and corporations are very cautious to switch to other technologies once becoming familiar with certain development paradigms as moving from one technology to another one could be a very time-consuming, costly and error-prone process. This is considered a barrier to Model-Driven Development as well [Selic2006, Selic2003].

Different technologies (e.g. Joomla [Joom2001], Ruby on Rails [Ruby2010] and Ajax [Garr2007]) assist web developers addressing the above-mentioned issues. For the most part, these technologies facilitate web development by automating or removing many low-level tasks such as User Interface (UI) development or the asynchronous service invocations required in interactive applications. By addressing such tasks, these technologies, reduce the time-to-market of web applications. Many of these technologies use automated code generation. Some technologies' ultimate goal is to raise the level of abstraction such that high-level descriptions could be automatically mapped to low-level design models or code. Examples are found in tools such as WebRatio [WebRat2008], NDT-Tool [Escal2003], OOH4RIA [OOH4RIA2011] and Microsoft Visual Web Developer [VisWeb2011]. Yet, these technologies cannot support web design models that could be reused for developing applications for more than one specific platform – here we define “specific platform” as a set of concrete technologies used for implementation. Developers often repeat the same type of tasks for every specific platform, whenever they redevelop an application, component or function. Approaches that can flexibly target several platforms from a single high-level description are, hence, still required in order to support the need to deploy applications on multiple platforms such as, for instance, desktop and mobile platforms.

Effort needs to be focused on minimizing the amount of work required to repeat the common tasks developers perform on every new platform using different technologies. Researchers and practitioners look for common patterns that map a certain design to specific code fragments. Once such mappings are identified, one can reuse them in every specific platform accordingly. Methods and techniques based on automated transformation of software models between different specific platforms are especially interesting in this area. Model-Driven Development (MDD) is a natural candidate as it provides means for generating software

¹ GWT has a Python-based version too.

applications starting from models. MDD allows transformations between different specific platforms and provides ways of automating parts of the process. Several MDD approaches have been suggested such as the ones by Stahl et al. [Stahl2006], Beydeda [Beydeda2005], and Hruby [Hruby2006].

The OMG approach known as Model-Driven Architecture (MDA) [MDA2003] introduces a terminology related to separation of concerns, with the concepts of CIM, PIM, and PSM as shown in Figure 1. A Computation-Independent Model (CIM) is at the most abstract level. It represents the context and the purpose of the modeled application without any computational specifics. A Platform-Independent Model (PIM) describes the behaviour and structure of the application regardless of the implementation platform(s). A Platform-Specific Model (PSM) contains the required information regarding concrete platform(s) that developers may use to implement the design and to obtain an executable application.

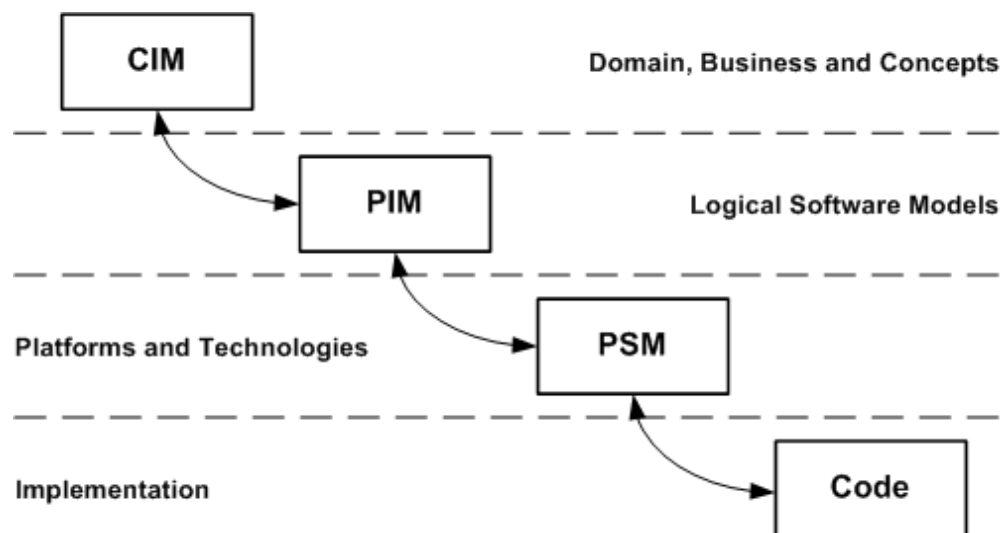


Figure 1 The MDA Framework

MDD has been applied to web development to a great extent – some examples, which will be discussed in this thesis, are [Ceri2006], [Famol2008-1], [Hern2010], [Kaval2007], [Nikol2005], [Schauer2007], [Wu2007], [Andro2007], [WebRat2008] and [Whit2004]. These approaches along with other tools and techniques – used to develop web applications following MDD principles – are referred to as model-driven web engineering (MDWE) approaches. A MDWE method requires the precise definition of web meta-models used as the source and the target of transformations. The existing MDWE approaches are tuned toward specific platforms

or develop only parts of web applications. These approaches generally use concrete meta-models adapted to their target specific platforms. Hence, the transformations are platform-specific and may not be used to develop the same application for other specific platforms.

1.2 Problem Statement

The ability of a model-driven method to target multiple platforms depends on the abstractness of the underlying model. Such an abstract model must allow the description of relevant features of Web applications independently from the peculiarities of specific platforms. In order to formally define transformations, this abstract model is used as the meta-model based on which the source and target models are designed. The abstract model needs to support clear and formal mapping rules between modeling elements. That is, for every modeling element, it must be clear to which other elements it should be mapped and by which mapping rules. Additionally, an abstract model for web applications should be defined to cope with the World Wide Web platform volatility. This thesis examines the possibility of designing such an abstract model.

1.3 Hypothesis

It is possible to create an abstract web model based on common features of the existing models and emerging web technologies so that this model can be mapped to virtually any web platform.

This is tested by implementing model-driven transformations that map the abstract model to different specific platforms. These transformations can be used to automate the process of creating platform-specific design models, and are hence called platform-specific transformations. Furthermore, the platform-specific transformations validate the abstract web model by verifying the capability of the model to be mapped to several specific platforms.

1.4 Solution Layout

Despite their variety, web technologies share many characteristics. These characteristics address web application structure, navigation paths, logic control mechanisms, data access techniques, user interface (UI) components and the principle of separation of concerns. More recent approaches such as Web 2.0 [O'R2005] have imposed additional necessities. Web 2.0 presents a different technological approach to client-side operations, which allows more flexible and

interactive UIs. The characteristics that enable such advancements should also be considered as a part of common web features. In this thesis, MDD is approached as a facilitating technique for examining the capabilities of a common abstract web model for specifying web applications and transforming them to multiple platforms.

Models of web applications that are built upon our abstract model belong to the Platform-Specific Models (PSM) level of MDA; we name this type of PSM, Abstract-Platform Specific Model (APSM). The APSM encompasses a set of modeling elements delimiting common necessities of web-based applications regardless of the platform on which the application is deployed. Such features form an abstract platform [Almei2004]. The APSM is platform-specific in the sense that it describes features specific to the web platform; it is also abstract since it does not contain details of specific web platforms but only their shared features. The term *APSM* is specifically used to name the abstract web model developed in this thesis; in order to denote instances of APSM we will use the term *apsm* throughout this thesis. We distinguish the APSM from Specific PSMs (SPSMs) that pertain to concrete web implementation platforms such as .Net or J2EE.

A smaller subset of the APSM is used for specifying the PIM. Instances of this PIM define the input to the implemented approach. The PIM is used to model a web application using a minimal set of graphical symbols and diagrams. The intent of this model is to prepare a venue for creating a utility for defining abstract web applications visually. In this thesis, the term *PIM* is used to refer to the meta-model, while the term *pim* denotes instances of PIM. The APSM is typically used, in this thesis, as the target meta-model for specifying transformations from *pims* to *apsms*. These PIM-to-APSM transformations are completely re-usable when for instance a web application needs to be ported to different concrete platforms. Only the transformations required for generating a specific PSM from the APSM need to be supplied; and only if this is the first time that SPSM is targeted. Since the APSM is semantically closer to a SPSM, this latter part requires less effort to develop in comparison to the conventional PIM-to-PSM transformations. In this thesis, transformations map the abstract model to specific platforms defined based on widely used technologies such as AndroMDA [Andro2007], WebRatio [WebRat2008], Google Web Toolkit (GWT) [GWT2010] and Microsoft .Net [Net2011]. In all

cases, the name of a specific platform in capital letters, e.g. *GWT*, means the meta-model, and the same name in small letters, e.g. *gwt*, means an instance of the meta-model².

The APSM has been designed following three principles:

- In order to ensure MDD transformations can run unambiguously, semantic associations between elements of different viewpoints are supplied as required.
- The APSM integrates functional requirements and aspects related to the user interface through state machines that model use cases, as well as UI prototypes. This allows modeling of the input models at an abstract level.
- The APSM integrates features required for Web 2.0 applications.

We have chosen to use an existing UML-based model as basis for our abstract web model. UML was selected, as it is an established modeling notation standard. We refer to the version 2.1.1 of UML [UML2007] in this work.

Figure 2 presents a high-level layout of the solution. As this figure shows, a *pim* is modelled using use cases, state machines and UI prototypes. This is mapped to an *apsm* using a general mapping, which is general in the sense that it results in a web model that is not specific to any specific platform. Specific mappings then map to specific platforms.

The APSM and its supporting transformations address the need for reusable models that could be mapped to multiple specific platforms. 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 data mappings as a part of the PIM. More importantly, the common features required to process data and communicate data objects between different layers and components are targeted.

Other benefits can be foreseen from our approach. By relieving developers from low-level platform-specific design, the approach has the potential to shift the development task to issues related to business needs. Another benefit is shortened development time. This could help web developers to overcome the problem of schedule delays, which is recognized as one of the

² This convention applies to the name of meta-models and their instances only. Modeling elements are not case-sensitive, i.e. a state named *Run State 1* is equivalent to a state named *run state 1* from the same state machine.

top five most-cited problems with large-scale web systems [Epner2000]. These benefits, however, need to be further validated in future work.

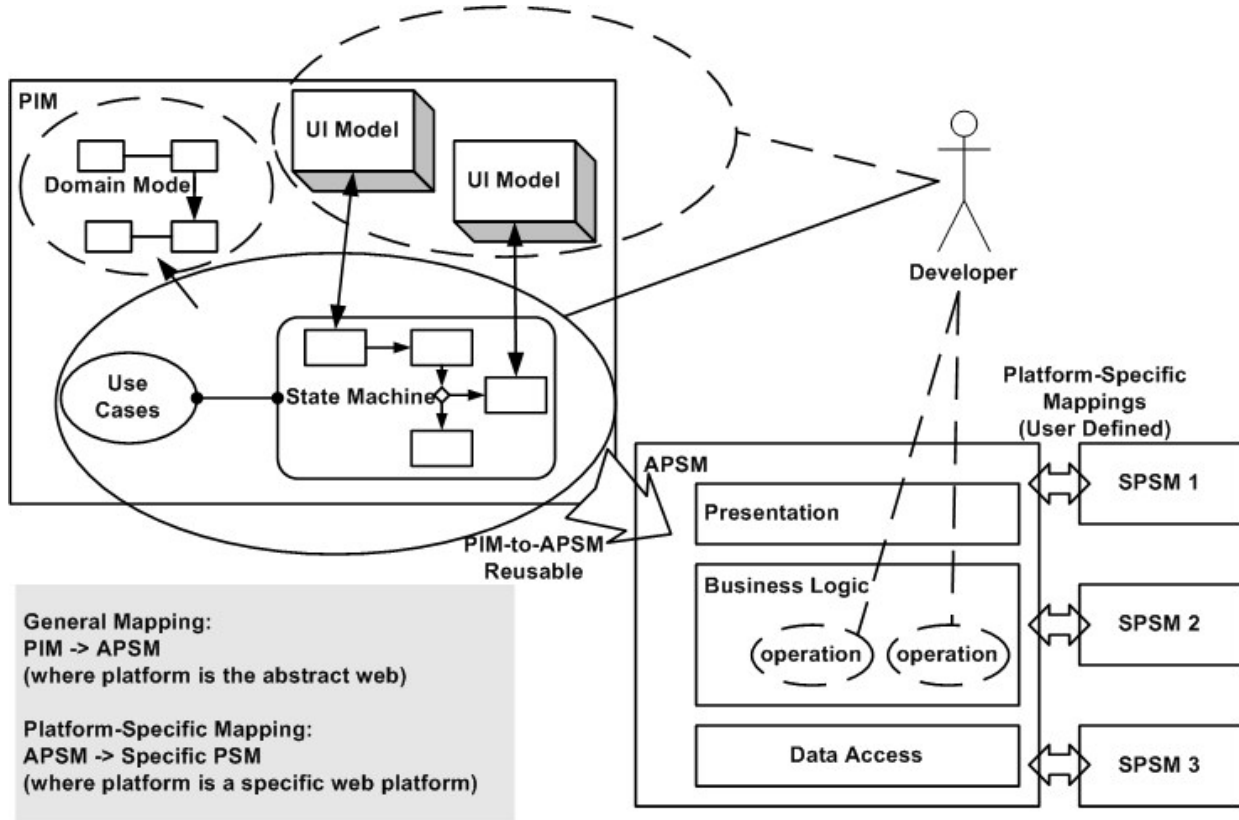


Figure 2 The Layout of a Solution based on the APSM

1.5 Research Methodology

We started by looking at different approaches that have the potential of mapping an abstract web model to multiple web platforms. For the most part, this question leads to MDD. The preliminary research led to the conclusion that the existing techniques are concerned with mapping a PIM to a PSM. Hence, a research question was stated as if there could be mappings to transform a PIM to multiple PSMs. Transforming a PIM to multiple PSMs necessitates the design of multiple platform-specific mappings. This would not provide much savings if platform-specific transformations are to be drawn from a PIM to every PSM separately. The idea of the APSM as a mediator between the PIM and the PSM to shorten the platform-specific part of the mappings was, hence, developed. The next step was to research the existing web models and technologies. This study resulted in a literature review that lists a set of features the APSM should possess.

The evaluation process started with defining a number of specific platforms as targets of the transformations. As web developers define a technology stack prior to developing an application, we defined every specific platform in terms of the technologies used to build and deploy the application. Creating sample applications based on different case studies was the next step. Our objective was to validate the hypothesis by mapping web applications to more than one platform. Therefore, we defined mappings required to support the APSM. The implementation of components was a part of the validation process. The mappings were implemented using QVT relations [QVT2008] and other utilities were developed to support the APSM using Java, Visio and VBA [Vis2010]. Finally, the results were formalized and documented. A model-driven approach based on the implemented components was also proposed.

1.6 Contributions

Several contributions have resulted from this thesis. These are: a survey of the existing models and technologies, the APSM, formal transformations implemented using Query-View-Transformation (QVT) [QVT2008] relations and intended to validate the abstractness of the APSM, several case studies, utilities implemented to facilitate the usage of the APSM concept, and a visual language for defining the input model. The main result is the APSM as an abstract web model. The APSM concept has been also applied to model-driven transformations targeting more than one specific platform. Following is a brief description of the contributions:

- **The APSM and comparison to other models:** A prerequisite for developing the concept of the APSM is an extensive survey on the existing web models, languages and technologies (Chapter 3). The results of this survey are formalized as a scoring table, which summarizes the existing research with respect to several common requirements of a web model (Section 3.6). An existing model [Botter2007] that scores higher in this table is selected as the basis of the APSM (Section 3.5). In order to support web applications in the context of MDD, the selected model is predominantly extended to form the APSM. The APSM is described as an abstract UML class model, which comprehensively supports several common structures of web applications (Chapter 4).
- **Implementation on various platforms:** The APSM is designed as an abstract web model that could be mapped to virtually any custom web platform. For

validation purposes, four specific platforms based on Google Web Toolkit, AndroMDA development kit, WebRatio Tool and Microsoft .Net have been selected, and model-driven transformations required to map the APSM to each of these specific platforms have been defined (Chapter 7). These transformations (Sections 7.1, 7.2, 7.2.2 and 7.4), implemented using QVT relations, are another contribution of this thesis while the meta-models specifying the four specific platforms are other minor contributions related to the QVT relations. These models are built and maintained using the Eclipse Modeling Framework (EMF) [EMF2009].

- **The PIM and its tool and language:** Defining a web application directly using the APSM would require detailed work. Using the APSM at this level of detail would hence be very difficult and error-prone. In order to facilitate defining a web application at an abstract level using the APSM, a smaller subset of its modeling elements is selected. This model, which is considered the input of the approach, serves as the PIM as well (Chapter 5). For presentation purposes, the PIM is topped-up with a visual language (Section 5.1). Web applications are defined using this visual language. Models built using this visual language are transformed to *pims* using Java utilities. A set of QVT relations is also provided to map the *pims* created based on this visual language to the APSM (Chapter 6). The PIM, its related visual language and the QVT relations that map it to the APSM are hence additional contributions of this thesis.
- **Case studies and utilities:** Several case studies have been conducted as part of this thesis research. Case studies served to validate our proposed approach and to be used as guidelines for using the approach. A number of utilities have been implemented in order to build these case studies and to set up a working environment for the research. These utilities are intended for converting the visual language files to the *pims* readable by the QVT relations. All together, these utilities and their accompanying case studies form the implementation package of this thesis (Chapter 8) and provide an environment for evaluating the approach (Chapter 9).

The following is a list of the publications that have resulted from this thesis:

- A. Fatolahi, S. S. Somé and T. C. Lethbridge. A Meta-Model for Model-Driven Web Development. In the International Journal of Software and Informatics, Special Issue on Software Modeling, Volume 6, Issue 2 (2012)
 - This paper discusses the merits of the APSM as an abstract web model and discusses the mappings from the APSM to example SPSMs [Fatol2012].
- A. Fatolahi, S. S. Somé and T. C. Lethbridge. Model-Driven Web Development for Multiple Platforms. Journal of Web Engineering, Vol. 10 (2011) No. 2. (pp. 109-152). June 2011.
 - This paper examines the APSM in the context of a model-driven process to develop web applications [Fatol2011-1].
- A. Fatolahi, S. S. Somé and T. C. Lethbridge. Toward Reusability In Web Modeling Using QVT Relations. In: Proceedings of the 7th International Conference of Web Information Systems and Technologies (WEBIST 2011), pp. 383-386. SciTePress 2011.
 - Here, we discuss the reusability features of the APSM and the mappings associated with it [Fatol2011-2].
- A. Fatolahi, S. S. Somé and T. C. Lethbridge, Designing a Map of Mappings: Visualization of QVT Relations using Basic Petri-Nets, In the Proceedings of the 2nd International Workshop on Future Trends of Model-Driven Development (FTMDD 2010), pages 33-45. INSTICC Press 2010.
 - A visual technique for designing QV relations is discussed in this paper, which is also used for depicting the transformations in some of the documents referenced in this thesis [Fatol2010-2].
- A. Fatolahi, S. S. Somé and T. C. Lethbridge. Towards A Semi-Automated Model-Driven Method for the Generation of Web-based Applications from Use Cases. In: Proceedings of the 4th Model Driven Web Engineering Workshop (MDWE2008) in MoDELS'2008. pp. 31-45. Published Online: <http://ceur-ws.org/Vol-389/>

- This paper discusses the first draft of a model-driven method using the APSM to develop web applications [Fatol2008-2].
- A. Fatolahi, S. S. Somé and T. C. Lethbridge. A Model-Driven Approach for the Semi-Automated Generation of Web-based Applications from Requirements. In: Proceedings of 2008 International Conference on Software Engineering and Knowledge Engineering. pp. 619-624. Knowledge Systems Institute Graduate School 2008.
 - A sample mapping from requirements to a specific Web platform was presented here [Fatol2008-1].
- A. Fatolahi, S. S. Somé and T. C. Lethbridge, Automated Generation of Abstract Web Models using QVT Relations, September 2010, Technical Report TR-2010-06, School of Information Technology and Engineering, University of Ottawa. Available from: www.site.uottawa.ca/eng/school/publications/techrep/2010
 - This document explains the basic idea of an abstract Web model and lays out the general structure of the QVT relations used to build the transformations [Fatol2010-1].
- A. Fatolahi, S. S. Somé and T. C. Lethbridge, A Meta-Model for Model-Driven Development of Web Applications, July 2010, Technical Report TR-2010-04, School of Information Technology and Engineering, University of Ottawa. Available from: www.site.uottawa.ca/eng/school/publications/techrep/2010
 - A technical explanation of the APSM with a case study [Fatol2010-5].
- A. Fatolahi, S. S. Somé and T. C. Lethbridge, Automated Generation of Abstract Web Applications using QVT Relations, August 2008, Technical Report TR-2008-02, School of Information Technology and Engineering, University of Ottawa. Available from: www.site.uottawa.ca/eng/school/publications/techrep/2008
 - This document explains the basic idea of an abstract Web model and lays out the general structure of the QVT relations used to build the transformations [Fatol2008-3].

1.7 Outline

This thesis is organized as follows. In Chapter 2, background and basic concepts are illustrated. Chapter 3 reviews the existing research work and technologies and lists the requirements of an abstract web model. Chapter 4 presents the APSM and its features. Chapter 5 explains PIM and its visual input language. In Chapter 6, mappings from PIM to APSM are described and transformations to run those mappings are listed. In Chapter 7, mappings from APSM to SPSMs are explained and transformations to run those mappings are listed. In Chapter 8, we provide a visual insight into the implemented approach in order to provide a detailed understanding of how to work with the APSM. Chapter 9 evaluates the APSM by discussing the implementation, methodology and results. Chapter 10 concludes this thesis. Finally, five appendices are as follows: Appendix A presents a detailed description of every class in the meta-model, Appendix B, Appendix C and Appendix D review two case studies using different approaches. Appendix E is a glossary of some specific terms of this thesis.

Chapter 2. BACKGROUND

In this thesis, we deal with several concepts coming from two different areas of software engineering discipline: Web-based application development, and model driven development (MDD). The two share some concepts such as the separation of concerns and the general terminology inherited from software engineering including design, modeling and implementation. In recent years, MDD has been intensively applied to web engineering leading to Model-Driven Web Engineering (MDWE) [MDWE2011]. MDWE deals with the application of abstract models, languages and automated transformations to web engineering. In this chapter, basic concepts of MDWE are elaborated and positioned from the viewpoint of this thesis. Other related approaches to MDWE are discussed in Chapter 3.

2.1 Separation of Concerns

Separation of concerns is the art of placing boundaries in between different aspects of software systems. The idea is to ensure every concern contains only attributes and behaviour that are essential for recognizing it and that the concern is independent of the attributes and the behaviours of others. As Greer [Greer2008] puts it “applying the principle of Separation of Concerns can increase the stability, maintainability, extensibility, reusability, adoption, and even the logical and aesthetic design and flow of a system”. The notion of separation of concerns has been in the realm of computer science for a few decades. According to Dijkstra [Dijk1982] “one is willing to study in depth an aspect of one's subject matter in isolation for the sake of its own consistency, all the time knowing that one is occupying oneself only with one of the aspects.” Many of the existing technologies and paradigms have been invented with the separation of concerns in mind. Examples range from HTML and CSS that separate content from style, to aspect-oriented programming that focuses on primary functions by separating common concerns such as security and authentication from more specific ones.

Separation of concerns may refer to horizontal or vertical separation. Vertical separation is mainly performed by dividing the application into modules that integrate the functionality related to the same features of the system (e.g. classes related to the same use case). Horizontal

separation refers to logical division of the features into layers that encompass items with closely related role³. Two common forms of this type of separation of concerns are shown in Figure 3.

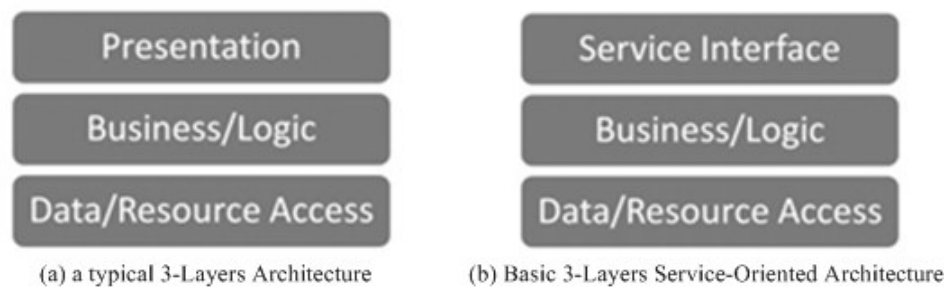


Figure 3 Examples of Horizontal separation of concerns

In this thesis, both vertical and horizontal separations are considered. Vertical separation is mainly focused on use cases. The APSM vertically separates elements of web applications by distinguishing each use case, assigning a use case controller and a state machine to the use case and creating dependencies from controllers to other required classes. The APSM also horizontally separates the concerns of an application into three main layers, which are presentation, logic and data access.

2.1.1 Model-View Controller (MVC)

Model-View Controller (MVC) [Reen1979] is an architectural pattern widely used in academia and industry for software development. MVC is based on the idea of separation of concerns in terms of separating the presentation of an application from what happens behind the scene as the core of the application. Later, MVC was extended to cover data access and service layer and since then it has been used in web development practices by most of the industrial frameworks such as .Net, J2EE, Spring and Struts [Shklar2003]. The main elements of this pattern are introduced in this section because it is widely referred to in the related studies, technologies and languages.

Figure 4 depicts a general framework for MVC. The *Model* contains domain-related data and logic of the application while the *View* component provides the presentation side. Once *Control* receives an event, it may apply changes to both the *Model* and *View*. Any change to the *Model* side may also result in an appropriate change in the *View* but changes in the *View* do not necessarily affect *Model* components [Reen1979].

³ See Appendix E for the definition of terms *Layer* and *Role* in this thesis.

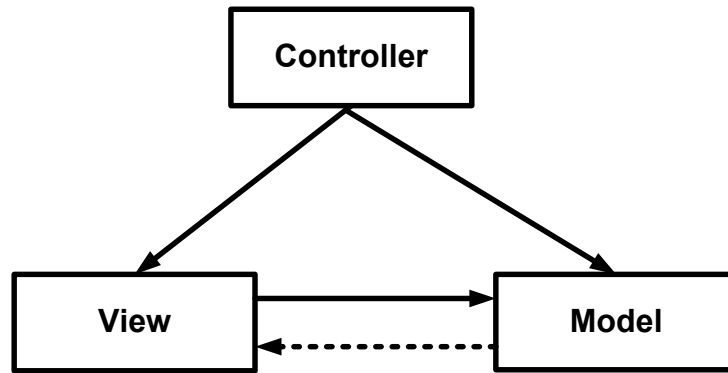


Figure 4 The Original MVC

In order to understand how an MVC-based architecture is applied to web applications, one needs to realize that one important issue when designing a web application is the separation of presentation and content [Shklar2003]. While the presentation shows the data and information, the presented content should not be confused as a part of the *View* component, since it belongs to the *Model* part. Nowadays, web application frameworks recognize the following as the constituents of a typical web information system:

- Presentation structure: this is the organization of presentation units, possible paths through them and their relationships (*View*).
- Presentation elements: these are what the user sees, touches and communicates with such as the buttons, texts and lists (*View*).
- Information: this includes the data that is used in the application as well as the processed information that is presented to the User (*Model*).
- Navigation: this is sometimes seen as a part of the data model that defines different views to data and possible navigations throughout the data model. For example, it would say which objects a typical table is composed of and what attributes of those objects shall be considered (*Model*). The term “navigation” is also used to refer to the map of the all possible paths a web application may go through (*View*).
- Behaviour: this is the logical model for processing and controlling. The behaviour specifies what occurs in response to events with suitable reflections within the information and presentation parts (*Control*). In many cases, the operations

working directly with data sources are considered a part of the *Model* rather than *Control*.

Although being a popular pattern, the application of MVC to web development has its own challenges. Some of the problems are due to the fact that MVC predates the modern web. As an example, Rossi and Schwabe [Ross2006] express that MVC lacks the support for structures of the rich Internet application frameworks such as Google Web Toolkit (GWT) [GWT2010] and .Net [Net2011]. The reason is, MVC is based on a purely transactional view of software applications and hence, cannot support many aspects of the modern Web such as web service invocation and Web 2.0 asynchronous calls. They also mention that MVC does not take into account the navigation aspects of an application. Another limitation is reported by Shklar and Rosen [Shklar2003] who noted that the presentation technologies for web applications are so complex that even MVC could not handle them. As a result, MVC-based web development frameworks cannot explicitly separate the design and programming concerns of web applications. The matter of proper separation of concerns in web applications is also largely affected by the developer's design decisions. Therefore, automated approaches – such as the one proposed in this thesis – that preserve a proper distinction between different concerns - are important.

According to our review of the MDWE literature, MVC is an influential underlying architecture for MDWE research. MVC is referred to in most of the related work as a reference model. Despite its weaknesses, the main idea of MVC to separate the presentation from control and main logic is still relevant. In this thesis, MVC is approached as an architectural pattern but more specifically the term *controller* is also applied as a design pattern [Larman2005].

2.2 Model Driven Development (MDD)

Software engineers have long used models for specifying software applications. A part of software engineers' job has therefore consisted in manually working from models to obtain executable code. Model-driven development tends to take this process further by allowing the modeling artefacts to be directly used toward creating the working application. This is mostly done by replacing the manual transformation process by an automated one.

2.2.1 Model-Driven Architecture (MDA)

MDA is an effort by the OMG to standardize model driven software development [Stahl2006]. It can be seen as a framework composed of three different layers of modeling. The topmost layer is the layer of Computation-Independent Models (CIM). CIM represents models that are valid in spite of the computational options. Then there is the layer of Platform-Independent Models (PIM). PIMs represent systems and software design and architecture; however, they do 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 specific platforms. Lower than PSM, there are Implementation-Specific Models⁴. These are real world objects and components, acting as a running version of the system. This thesis adds the Abstract-Platform Specific Model (APSM), an intermediate between the PIM and the PSM. The APSM is introduced in Section 2.2.3 and elaborated in details in Chapter 4.

The Meta Object Facility (MOF) [MOF2007] is the heart of MDA. The MOF provides a means for building new modeling languages or transforming models from different languages to each other. The MOF is composed of a small and concise set of elements to describe other modeling languages. Basic UML class modeling notations can be used to depict MOF-based models with few considerations [Frank2003-1].

MDA admits two levels of MOF-based languages [Frank2003-2]. The first level addresses languages rooted in the MOF itself. The best example of a first level language is UML. The second level involves different UML extensions such as a UML profile for Graphical User Interface GUI layouts [Blank2004-1] or extensions to other languages based on MOF. The APSM is built using this latter level of MOF-based languages. In order to facilitate model exchange among different tools and standards, an XML Metadata Interchange (XMI) format is also considered as a language pertinent to MDA [XMI2007]. Instances of the APSM and other meta-models in this thesis are implemented following the general guidelines of XMI.

MDA can be better understood by its meta-modeling mechanism, which is reflected in Table 1. While a model can be seen as an abstraction of a real-world domain, a meta-model is understood as an abstraction of the model itself. Hence, models defined based on a meta-model

⁴ Not an official MDA term.

have to conform to it. MOF is considered as the most abstract level of modeling in MDA named M3; this is also known as the meta-meta-modeling level because MOF is often used as a language for defining meta-models. Models defined based on MOF such as the Common Warehouse Model (CWM), UML and its profiles form the M2 level; these are meta-models. M1 contains models as designed in software engineering projects. These models are typically based on M2 models such as UML or its profiles. Finally, M0 contains real-world objects that are runtime instances of the elements found in M1.

Table 1. MDA meta-modeling mechanism

M3 (MOF)	Meta-meta-modeling layer, including the most abstract materials required to build new languages and interoperability standards
M2 (UML, CWM, ...)	Meta-modeling layer, providing the notation and formalism that can be used to model specific domains and systems such as UML profiles. This layer is fed by M3.
M1 (User Model)	Projections of M2 in terms of specific projects. This includes instances of M2-level models used for the specification of software applications.
M0 (Runtime Model)	Running versions of M1 including runtime objects.

In this thesis, we use several meta-models (M2) and deal with their instances (M1) in terms of various examples and case studies. The object models created when executing these examples belong to the M0 level. As a general rule, whenever we use capital letters we mean the meta-model but when we name a model using small letters we mean an instance of the meta-model. For example, *APSM* refers to the abstract PSM as a meta-model whereas *apsm* refers to an instance of the APSM.

The most important MDA standard used in this thesis is the Query/View/Transformation (QVT) relations. QVT relations are used to define transformations and are elaborated in Section 2.6.

2.2.2 PIM and PSM

A *pim* includes a collection of abstract elements for modeling a software system independently from specific platforms. A *psm* contains models that are customized with respect to the specific platforms and technologies the application will be built on. Both *pim* and *psm* are defined based on meta-models (respectively PIM and PSM). Since transformations are mostly defined from

PIM to PSM, it could be said that the PSM is partially affected by the PIM. The PSM is also affected by the chosen platform(s).

Use cases and domain models are common ways of expressing *pims* in the literature. However, the type of model used for describing the behaviour of use cases varies. Li et al. [Li2000] suggest no explicit behaviour model as part of their PIM/PSM as their model is a general approach to web application architectures. Similarly, Meliá and Gómez [Meliá2005-2] do not use a specific behavioural model, as their approach deals with configuration issues and does not give a special care to the contents of the functional model. A number of approaches based on UML-based Web Engineering UWE [Kroiß2008] use activity diagrams to describe the behaviour of use cases in their PIM. This could be considered comparable to using state machines because UWE was first introduced and used by Koch [Koch2000] when state-activity diagrams were still a part of UML⁵. The use cases are said to be supported by interaction diagrams by Rossi and Schwabe [Ross2006] but what the authors present as interaction diagrams are indeed state machines that depict the transitions from one web page to another one in both PIM and PSM. Baresi et al. [Baresi2006] suggest a state diagram in their W2000 model that is essentially composed of several states; this model is used to specify the required behaviour. Wu et al. [Wu2007] use textual use case descriptions in the PIM that are eventually transformed to interaction diagrams in the PSM. Distanto et al. [Dist2007] suggest sequence diagrams as part of their MVC-based logical model to build the PIM.

In this thesis, a single abstract model is proposed as the meta-model of both *pim* and *apsm*. A subset of the abstract model is used to define *pims* and serves as the input to the implemented process. The *apsms*, on the other hand, are modelled using the entire abstract model. Our PIM is composed of the following elements:

- Use cases modeled using a controller class as well as the include/extend relationships between use cases.
- State machines that specify the status of use case controllers as flow of the web applications with regard to sequences of web pages and other presentation units such as forms. According to OMG [UML2007], use cases are one type of *Behavioured Classifiers*. A *Behavioured* classifier is a specific kind of a classifier

⁵ The unified *State-Activity Diagram* from UML versions prior to 2.0 is now obsolete. Newer versions of the UML distinguish State machines and Activity diagrams.

that could have an associated *Behaviour*. Among the types of behaviours introduced in UML, we selected state machines to describe use case behaviours.

- Domain model in terms of classes, their attributes and associations.
- User Interface (UI). This is one of the distinguishing features of our approach. UI templates specify the desired user interface plus the information elements related to them. The specification of the UI templates must be provided by the developer.
- Abstract operations. These are operations assigning different parts of the UI to each other or to the information elements in order to define the logic required to perform client requests, create output and process the input. Examples of abstract operations managed in this thesis are sending emails, populating UI components, data-related operations, and (re)setting session/page variables.

The APSM includes:

- State machines enriched with states handling logic and behaviour, transition events, operation calls and parameters. These state machines form the behaviour of the application,
- Controller classes created per each use case and their operations.
- Domain classes and their equivalent information elements used for information storage and retrieval, and to support the UI components.
- Service classes handling data-related operations.
- UI model and the components.

State machines are at the heart of the APSM. This is in accordance with the existing practice. As mentioned earlier, in most of the existing work, state machines or similar models are used for describing the behaviour of web applications. Additionally, it is convenient to model a web-based application using state machines as one can easily map the application to a state machine by assigning one state to each web page or an equivalent presentation unit.

We opted out from using interaction diagrams. In order to use interaction diagrams, one needs objects that represent presentation and processing units. One would then express events and transitions in terms of object operations. It would be difficult to use interaction diagrams,

since our approach starts at the PIM level, where the operations of objects have not been modeled.

We also opted out from employing activity diagram. One should note that the behaviour of use cases is not directly modelled in this thesis but it is the use case controllers that are modelled. Further, we needed a diagram that shows the status of web application based on the status of the current controller thread. It is critical that every step in the diagram represent the status of application in terms of being busy doing some task or presenting information and input forms. Also, the behaviour diagram needs to be capable of responding to external events. Activity diagrams are more suitable for modelling things such as use cases, where it is not necessary to assign the responsibility of each step to specific classes. It is also common to use activity diagram to model business processes or very low-level tasks that do not need to interact with the outside world after launch. One might find our PIM state machines very similar to activity diagrams but at the APSM level such similarity is not found.

2.2.3 Platform

As Kleppe et al. [Klep2003] put it, a platform is “a set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns, which any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented”. A platform model is a conceptual definition of the features and services available through a platform [Klep2003]. Defining a platform, hence, requires selecting a set of technologies, features and services as well as deciding the level of abstraction. This is to ensure that the platform model is maintained at the desired level to avoid unnecessary expansion of details about the functionality and structure of the applications developed based on the platform.

In practice, platforms are formed as a set of technologies that are used in customized configurations. Such platforms are referred to as “specific platform” in this thesis. The term, platform is also used to simply refer to an industrial standard such as Java or .Net. These types of platforms are merely referred to as “platform” in this thesis. The contributions and results of this thesis will be evaluated against its potential to advance reusability and portability of web application designs mainly with respect to specific platforms.

Abstract Platform

The notion of platform as illustrated by OMG in the MDA official document [MDA2003] is a relative concept that is dependent of the context and the software engineering activity. Thus, the contents and the semantics of a PIM and a PSM must be understood upon the definition of specific platforms. For example, a user interface model created in accordance with the *Spring* framework [Sprin2008], shall be considered platform-independent when the developer's task is to model the required data mechanisms to feed the UI model from a MySQL database. Such UI model does not rely on peculiarities of any specific database, and since the context of the activity is concerned with databases, the platform is defined based on MySQL, from which the *Spring* framework is independent.

This is one of the differences brought by MDA in comparison to conventional development methods, where the concept of platform is understood as rather fixed and mean a set of specific implementation targets. In MDA however, the term *platform* could be applied both to a single specific technology while ignoring the details of others or a combination of several technologies used for different purposes. As another example, a web-based application modeled using the *Struts* framework [Strut2010], as a UI technology, remains free of the database technology. In this case, the model is platform-specific if we define the platform as the *Struts* framework but platform-independent if we identify the platform as one of the database management systems. In other word, a MDA model could be platform-specific against one platform while being platform-independent compared to another platform at the same time.

In this thesis, an intermediate level termed Abstract-Platform Specific Model (APSM) is added to the three conventional layers of MDA. The APSM, an intermediate between the PIM and PSM in the context of Web applications, is concerned with models specified with respect to common features of different web platforms. Such features form an abstract platform [Almei2004]. The APSM is platform-specific in the sense that it describes features specific to the abstract web platform; it is also abstract since it does not contain details of specific web platforms but only their shared features.

A few of the existing works have examined an approach similar to the APSM. He et al. [He2005] recognize the relative nature of platform and suggest a similar approach. However, their discussion remains at a conceptual level; that is, the authors do not suggest the details of an

abstract web model. Ubiquitous Web Application (UWA) [UWA2002] is another example of an abstract framework for defining an abstract web application. UWA is suitable as a reference framework that one might use to build a method or a model. Muller et al.'s approach [Mull2005] 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 thesis. The key to using Muller et al.'s approach would be a proper distinction between platform and its constituent technologies. A J2EE framework for model-driven development of web applications based on a universal web model with a focus on authentication features is described in Sakowicz et al.'s method [Sack2007]. These approaches are discussed in more details in Chapter 3.

2.3 Use Cases

According to OMG [UML2007], “a use case is the specification of a set of actions performed by a system, which yields an observable result that is typically of value for one or more actors or other stakeholders of the system”. Use cases are means for the documentation of what the system is supposed to do. A use case describes the sequence of interactions between actors and a subject system, in order to achieve a goal.

Cockburn [Coc2001] defines three levels of use cases that are:

- Summary-goal level use cases are used as a context to fulfill a scenario composed of multiple user-goal level use cases. In this thesis, these are only used to describe examples of using the APSM for describing multi-level designs.
- User-goal level use cases are the regular way of describing scenarios a primary actor will take to fulfill a goal. These are common in this thesis.
- Subfunction-level use cases are only necessary for readability purposes or in circumstances that a certain piece of functionality is repeatedly called by several use cases. These may be used for expressing low-level functionality and are not considered in this thesis.

UML 2.0 allows designers to describe the behaviour of a use case using textual descriptions, state machines, and activity or interaction diagrams. Other non-UML methods have been also proposed such as use case maps [UCM2008] that are used for modeling causal paths

that cut through different organizational units. In this thesis, state machines are used for specifying the behaviour of use cases. Although textual description of use cases is the most frequent way of describing use case behaviours [Larman2005], [Phalp2007], the usage of state machines should not be considered a drawback. Several tools and methods for automated synthesis of state machines from textual description of use cases have been reported by different researchers [Behr2002], [Somé2006] and [Sut1998]; UCed [UCed2005] is an example of a tool to perform such tasks.

Listing 1 is a typical login use case description in natural language text. In this use case, a User enters a username and a password. The system verifies the username and password; if correct, it shows a home page. If the username and/or password are not correct and the User has not reached the maximum unsuccessful login attempts, the system returns to the login page with an error message. However, if the maximum number of unsuccessful attempts is reached, the account will be locked and the user can select to contact administrator for further assistance, which is a separate use case.

Listing 1 A typical login Use Case

```
Name: login
Actor: User
Precondition: System is up
Steps:
    1- User enters username and password
    2- System verifies username and password
    3- System shows welcome homepage
Alternatives:
    2-a- username and/or password not correct
        2-a-1- System shows an error message
        2-a-2- goto 1
    2-b- login not approved and maximum unsuccessful login attempts reached
        2-b-1 System shows account locked message
        2-b-2- include contact administrator use case
```

A use case model is a collection of use cases, their relationships and the relationships they might have with actors. Actors usually participate in use cases in order to perform a task. Use cases can include or extend other use cases. The *include* relationship denotes that a use case includes the functionality of another use case. The included use cases could have been fragmented from the including use case in order to improve readability or because of the fact that it could be an individual use case by itself that an actor deals with on certain occasions. The

extend relationship stresses that an extension use case augments the behaviour of an original use case to some extent. The extension can consist in adding more steps, specializing the way a certain step occurs or providing different responses under alternative conditions.

A use case diagram is used to show the use cases, their relationships and the actors communicating with those use cases. Figure 5 shows an example use case diagram. As this figure shows, Customer makes an order in communication with Seller. The *View Catalog* use case extends the functionality of the *Make Order* use case by giving the option to the Customer to view the catalogue. Finally, the *Make Order* use case includes the *Make Payment* use case.

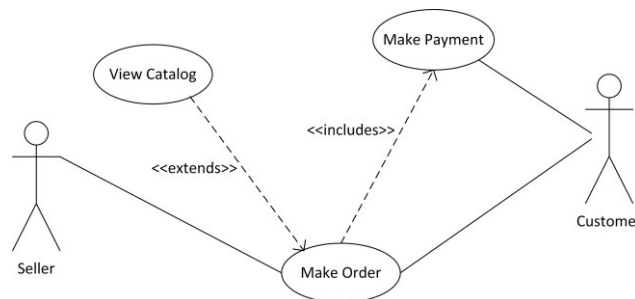


Figure 5 A Use Case Diagram

2.4 State Machines

State machines are used to show the dynamic behaviour of things such as use cases, transactions and objects in terms of *states* and *transitions*. A *state* specifies a set of conditions and a *transition* relates two states when an *event* occurs. We use UML state machines in this thesis. OMG [UML2007] defines two types of state machines in the UML superstructure:

- *Behavioural state machines*: These state machines can be used to specify the behaviour of various modeling elements. For example, they can be used to model the behaviour of individual entities (e.g., class instances). Behavioural state machines are also used for modeling the behaviour of use cases.
- *Protocol State machines*: Protocol state machines are used to express a system entity usage protocol. According to OMG [UML2007], “Protocol state machines express the legal transitions that a classifier can trigger. The state machine notation is a convenient way to define an object lifecycle, or an order of the invocation of its operations. Protocol state machines do not preclude any specific

behavioural implementation; instead they enforce the legal usage scenarios of classifiers, interfaces, and ports”.

A protocol state machine prescribes general circumstances under which an object may function but a behaviour state machine elaborates a specific possible execution such as, for example a flow of events imposed by a use case. Syntactically, behavioural state machines can contain operation details such as transition actions whereas protocol state machine are only allowed to refer to pre/post conditions.

The state machines used in this thesis, are behavioural state machines. The reason is that we need to describe flow of events in response to the User’s requests, and behavioural state machines are suited for that. UML state machines are composed of several elements among which the following are pertinent to our work:

- *States*, which we use to represent the status of web applications.
- *Choices*, that are used for building conditional flow of events especially for representing alternative use case steps and use case extensions⁶.
- *Transitions*, used to express the change of status in web applications. Transitions in this thesis may carry events and guards.
- *Initial and Final states*, used to show use case sequencing. A use case final state may be connected to another use case initial state to specify a sequential relationship where one use case executes after the other.
- *Sub-state machines*, used to describe states in detail as state machines.

The details of the APSM state machines are essentially based on what appears in use cases as discussed in Section 2.3. In this thesis, states are generally used for representing different presentation and processing steps. Transitions either represent the occurrence of an event or the result of an operation. The latter are often used in combination with choice nodes. At the PIM level, we only consider states and transitions. Operations, events and parameters are automatically added at the APSM level. An example is provided in Figure 6. This state machine describes the behaviour of the login use case shown in Listing 1.

⁶ These are dynamic decision points.

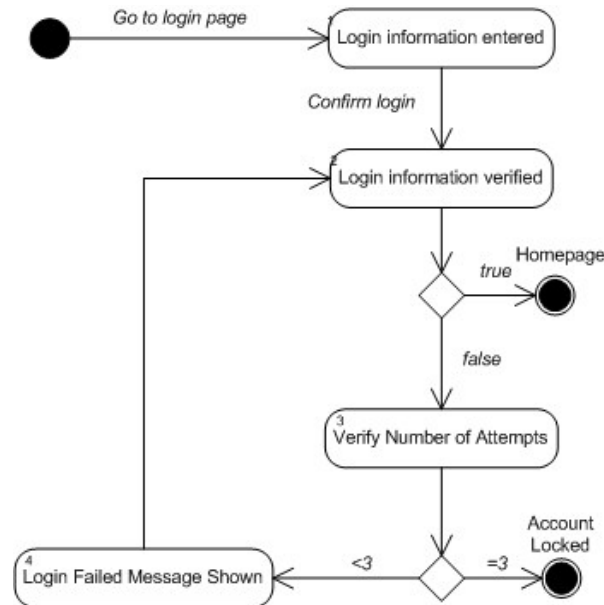


Figure 6 Login state machine

2.5 Information Systems

There are different categories of web applications such as search engines, news groups, online communities, weblogs, online media sites and forums. The applications targeted by our approach can be categorized as web information systems. Information systems have long been subject of patterns, best practices and standards and hence, are suitable to be examined in the context of this thesis. The term, *information system* has been applied to a variety of computer applications. In this thesis, an information system is seen as a system whose main functionality is related to data accessing, data processing and data recording in an organized data store, with an extensive interaction with users.

This type of application could also be referred to as *data-centric web applications*. However, the term *information system* is routinely used in the literature to refer to systems where the main functionality is data processing. For example, Alter [Alter1996] defines an information system as “a system that uses information technology to capture, transmit, store, retrieve, manipulate, or display information used in one or more business processes”. Kaseh et al. [Kasc2004] accept the same definition as a functional description of an information system by generalizing the concept of information or data as linguistic expressions. Kaseh et al. [Kasc2004] also use a structural perspective, in which an information system is composed of several components including people, process and data, where the technology is only one

component in this collection. The term *information system* in this thesis refers to the technology part or the software application that performs the information-related tasks.

Information systems have been in use for a long time – they predate the web itself by far – there is a deep understanding of the ideas behind the development of such systems. Information systems are mainly used to record, process and browse data. Four basic capabilities are defined for them, using the acronym “CRUD”; Create, Read, Update and Delete:

1. Creating a new record of data
2. Reading a set of data instances according to a defined criteria
3. Updating specific elements of data according to certain criteria and a change pattern
4. Deleting an existing data object

Applications with CRUD capabilities are specifically interesting from an MDD point of view [McNei2004] as their functionality is straightforward. The CRUD code can be generated with the least intervention required from developers. Therefore, CRUD operations form a well-defined basis for most MDD methods and receive special attention in the APSM as well.

2.6 Transformations

A key aspect of MDD is the capability of model transformation. As Pierantonio et al. [Pier2007] put it transformations are the chaining feature that makes the automation process possible. An MDA transformation is defined as an execution of a mapping. The OMG [MDA2003] defines a mapping as a specification describing how to transform a PIM to PSM. This essentially means that a mapping is a platform-specific notion – according to OMG [MDA2003] - and the characteristics of a mapping are decided according to selected platform(s).

There are alternative definitions of model transformations:

1. Appukuttan et al. [Appuk2003] discuss two types of transformations: Relations and Mappings. Relations are bi-directional and non-executable while mappings are unidirectional and operational. Relations are expressed using OCL but mappings are based on the Action Semantic language. This suggestion is in fact a part of one of the proposals made in response to the QVT Request-for-Proposal (RFP).

2. Mellor et al. [Mellor2004] do not mention transformations or relations. They instead define mapping functions as a mechanism that defines how a particular mapping works, while a mapping is an actual execution of the mapping function in order to transform one model to another.

In this thesis, the term *mapping* is understood as an informal specification of a source-to-target pairs of relationships that bind modeling elements in a source and target models. A transformation is a formal implementation of a mapping and relations are used as an approach to define transformations.

2.6.1 Transformation Languages

Several approaches for expressing transformations of models exist. ATL [ATL2005] is a transformation language and tool, which uses a rule-based semantics with a small syntax. Caramazana et al. [Caram2004] suggest defining MDA transformations using UML collaborations. The idea is to assign collaborations to both PIM and PSM and define a collaboration model between the two. Scheidgen [Scheid2006] presents a transformation implementation language, Model Pattern (Mopa) that is an extension of the Java programming language. Mopa provides a pattern configuration facility to define possible configurations within the PIM and to match it to the target model. The transformation is less expressive, less specific [Scheid2006] and hence more flexible since programmers can play with all the aspects freely; but a lot of low-level work is required to generate the transformations as most of the setup work is left to the programmer. Marschall and Braun [Mars2003] introduce Bidirectional Object oriented Transformation Language (BOTL) to support transformations in the context of MDA. BOTL is a trade-off between algorithmic and graph languages. Akehurst and Kent [Akeh2002] present a relational language to define transformations from an abstract syntax to a concrete syntax with the help of the semantic domain. Varró and Balogh [Varró2007] describe an approach to automate the process of model transformation by example [Varró2006] using inductive logic programming. Yet Another Transformation Language (YATL) [Patra2004] is based on OCL. It uses a Java-like syntax to generate transformations⁷.

The eXtensible Stylesheet Language Transformation, XSLT [XSLT2011] is a language that can be used to represent XMI to XMI transformations. XSLT is also capable of representing

⁷ Kermeta (www.kermeta.org) is another alternative, which was not available when the research was in-progress.

transformations from one XMI document to another type of document and so could be used in the context of this thesis. Some approaches and tools such as WebTE [WebTE2008] use XSLT as the basis for the specification of transformations. XSLT documents may sound good for transferring the specification of transformations from tools to tools but, as a language to express transformations, the lack of a readable presentation and their verbose nature make the transformations a long-winded and fault-prone task [Mars2003].

In this thesis, we use QVT relations, an MDA standard. It should be noted that most of the above-mentioned approaches to transformations are based on responses made to the QVT RFP. The QVT standard can therefore be considered as the outcome of the RFP. Additionally, due to the lack of maturity of the topic of model transformations, most of the tutorials introducing tools and approaches to transformations cover very basic examples with few guidelines for building strong transformations for complex projects. The selection of QVT offers the advantage that, being an OMG standard, there is likely to be more efforts toward it with a better likelihood that flaws and challenges are discovered and resolved.

According to the OMG [QVT2008], there are two ways of expressing transformations: *operations* and *relations*. *Operations* provide a procedural approach for expressing transformations while *relations* transformations are declarative. Both *relations* and *operations* transformations are equally powerful and the final selection is only affected by the skills and interests of the developer as well as availability of tool support. The *operations* language has received attention mainly because it tends to be similar to the conventional programming languages. The *relations* language, however, has the advantage of making the development of bi-directional transformations easier [Steve2007].

Relations transformations may be expressed using a core or a relation language. The core language supports flat object pattern matching. As a result, a transformation expressed using the core language is more verbose. In this thesis, the relations language is used. Listing 2 shows the syntax of a QVT Relation.

Listing 2 QVT Relations Syntax [QVT2008]

```
Relation R
{
    Var <R_variable_set>

    [checkonly | enforce] Domain:<typed_model_1><domain_1_variable_set>
```

```

{
    <domain_1_pattern> [<domain_1_condition>]
}

[checkonly | enforce] Domain:< typed_model_n>
<domain_n_variable_set>
{
    <domain_n_pattern> [<domain_n_condition>]
}

[when <when_variable_set> <when_condition>]

[where <where_condition>]
}

```

A transformation is defined as a set of QVT relations that maps two (or more) models. The syntax of a relation R that transforms instances of *typed_model_1* to instances of *typed_model_n*, referred to as the source and target domains respectively, is shown in Listing 2. The clause, *when* defines a precondition that must hold before the matching of the source and the target. Respectively, the ‘*where*’ clause states a post-condition. The patterns used for defining source and target domains act as preconditions as well; that is, in order to run a transformation from *domain_1* to *domain_2*, a prerequisite would be a successful pattern matching of *domain_1_pattern* as the source element within the source domain *domain_1*. A domain is any distinguished typed element defined using a pattern; a domain matches a given modeling type in one of the candidate models of the transformation. A QVT relation may be a top relation; that is the validation of the transformation starts with such a relation. Non-top relations are only required to be validated when the relation invoking them, as a part of their *when* or *where* sections, is being validated. Listing 3 shows an example of *ClassToTable* relation [QVT2008].

Listing 3 Class to Table QVT relation [QVT2008]

```

top relation ClassToTable // map each persistent class to a table
{
    cn, prefix: String;

    checkonly domain uml c:Class {namespace=p:Package {},
                                   kind='Persistent', name=cn};

    enforce domain rdbms t:Table {schema=s:Schema {}, name=cn,
                                   column=c1:Column {name=cn+'_tid', type='NUMBER'},
                                   key=k:Key {name=cn+'_pk', column=c1}};
}

```

```

when {
    PackageToSchema(p, s);
}

where {
    prefix = '';
    AttributeToColumn(c, t, prefix);
}
}

```

Listing 3 describes a relation mapping a class to a table. The transformation proceeds from classes to tables. In order for the transformation to be valid, it is necessary that the *PackageToSchema* relation hold. This relation maps a package to a database schema, meaning that the table must belong to the schema generated as the result of mapping the owning package of the source class; such a mapping is required to guarantee the uniqueness of the created tables. The object pattern defined for the domain *c* indicates that the transformation looks for classes within the source model that are persistent and have a name. The transformation then maps every single such class to a database table with the same name as the source classes and a default column as the primary key. The transformation is afterward enabled by the validation of another relation that maps the attributes of the class to columns of the table.

2.6.2 Visual Transformation Languages

Complex sets of mappings such as the ones in this thesis are very hard to understand, follow and present in textual format. Several transformation visualization techniques have been proposed to help design, trace and understand model transformations. A large number of these approaches are based on Petri-Nets [Peter1981] but other approaches have been also suggested. In recent years, the trend has noticeably moved from presenting new visual transformation languages toward proposing visualization techniques for the existing transformation languages such as QVT relations. Examples are [Wimm2009-4], [Reit2007], [Kusel2009] and [Fatol2010-2]. We have visualized our transformations using different existing techniques [Fatol2010-1, Fatol2010-4]. However, the results have not been particularly useful especially with the large sets of transformations found in this thesis. They are not able to conveniently browse through complex transformations that happen at different levels and are very deep in terms of the level of transformation hierarchy mainly because of their rich notations that disallows the presentation of big transformations in limited space.

2.6.3 Types of Transformations

There are different types of mappings and transformations. According to Mellor et al. [Mellor2004] a mapping could be:

1. *Refining*, where the target meta-model has additional elements, compared to the source meta-model, such as a mapping from an analysis to a design model,
2. *Abstracting* that reduces the amount of details of the source model and raises the level of abstraction at the target,
3. *Representing*, this maps a language to its presentation, for example it can map a meta-model like UML to a representation language such as the UML notation,
4. *Migrating* that reorganizes a model to conform to another format such as data conversions, or
5. *Merging*, where multiple source models are weaved together to result in a single target model.

Refining and *abstracting* mappings are vertical mappings that change the level of abstraction but *representing* and *migrating* mappings do not change the level of abstraction and can be called horizontal mappings. Merging mappings can be vertical, horizontal or a mix of both.

With respect to Mellor et al.'s classification [Mellor2004], transformations in this thesis belong to refining mappings since they map PIM to APSM and the same meta-model is used for both the PIM and the APSM. Hence, transformations essentially add more details to the original *pim* to obtain a relevant *apsm*. One could also see this method from another point of view, based on which the method reformats a source model to generate a target model that has more details. All those new details could be seen as the result of transformations over the existing parts of the source model. Thus, some of the mappings provided in this thesis also show characteristics of migrating mappings as well. This is especially true regarding our platform-specific transformations that transform the APSM to non-UML targets such as WebML [Ceri2000].

It should also be mentioned that OMG [MDA2003] distinguishes two types of mappings in the MDA official document. One is the model-type mappings that transform PIM to PSM according to the types defined in the meta-model. Another kind of mappings is model-instance mappings in which specific transformations for each element according to the chosen platform

are taken. The *PIM-to-APSM* mapping in this thesis is a model type mapping but if one considers the whole set of *APSM-to-SPSMs* transformations as a single mapping function, it could be said that the latter is a model instance mapping.

2.6.4 Transformation Implementation Tools

There are various tools for QVT implementation. For this thesis, we needed a tool that supports implementing QVT relations and provides an environment for recording, debugging and running QVT relations. Integration with popular tools such as *Eclipse* was also considered a bonus at the beginning of this research. The following tools were available at the beginning of this work:

- Modelware smartQVT [Smart2008] is an implementation of QVT, which supports the QVT operations language.
- MomentQVT [Moment2008] is a QVT relations implementation that supports a command-line environment. MomentQVT is difficult for debugging and tracing transformations as it does not provide a visual interface for that purpose. Also, setting up the environment requires many parameters, which are not documented.
- An Eclipse plug-in is also available for QVT. It just provides the modeling elements required to build the relations but not the details of those relations.
- In this thesis, mediniQVT [Medin2008] is used as the tool for implementing QVT relations. This tool can be integrated with Eclipse by importing and exporting meta-models as well as source/target models using Eclipse-Modeling Framework (EMF) [EMF2009].

2.7 User Interface Modeling

According to Galitz [Galit2007], UI Modeling belongs to a larger field of computer science, Human-Computer Interaction (HCI). HCI addresses the communication between humans and computers in order to maximize the benefits of the humans. UI modeling has a direct impact on the success of a software application. According to Stephanidis [Steph2001] and Galitz [Galit2007], the programming efforts dedicated to user interface development in software projects exceeds 50% in many cases. UI modeling is a subset of HCI that addresses the design and analysis of the specific components the User can see, touch or hear when communicating with computers in order to provide inputs and receive outputs. HCI covers several other aspects

such as people's expectations, physical limitations, and psychological effects. In this thesis, UI modeling is supported by the meta-model and UI elements are defined as a part of the PIM.

Different types of UI models have been in use since the invention of computers. Most of the early computer systems had a command line or textual UI. These types of interaction mechanisms have nowadays been supplemented by graphical UIs (GUI) and web-based UIs. Although Web UIs and GUIs were originally different in terms their navigational and structural models, they have become very similar in recent years. This is partly due to the introduction of Rich Internet Applications (RIA), and the addition of web-specific styles to GUIs.

UI modeling has been enriched with the advent of Web 2.0. Web 2.0 [O'R2005] has enabled capabilities to share, interact and search information through text, video and image on the web. Web 2.0 can be seen as a wave bringing both technological and social changes. Several services based on Web 2.0 such as Facebook, Google and Twitter have become popular in a relatively shorter time than their similar predecessors that were defined in conventional web era, e.g. MySpace, Orkut and Yahoo. It is, therefore, becoming more and more important for web applications to move toward capabilities of Web 2.0. Web 2.0 introduces new ways of communicating with the User. The biggest shift is a shift to more interactive content that allows users to participate in discussions, leave comments and rank items in real-time. To this end, different units on the page must be independently updatable. In a pre-web 2.0 application a whole page would need to be refreshed either by the User or the application in order to update a part of it⁸. Web 2.0 allows only specific parts of a part to be updated asynchronously by facilitating behind-the-scene communication with server. These features are taken into consideration in this thesis.

There is a close relationship between UI and prototyping. A prototype is “a primitive version of a system” that is intended to be used for discovering critical issues as early as possible [Laues2005]. A very important part of prototyping is UI modeling that also affects the usability test planning. According to Lauesen [Laues2005], there are four different types of UI prototypes:

- *Hand-drawn mock-up*. The UI is modeled on a piece of paper. The behaviour of UI elements is also illustrated manually.

⁸ Alternatives include using hidden forms and frames or similar techniques.

- *Tool-drawn mock-up*. The UI is drawn using a tool but the tool is not a part of the development process. The prototype is then printed or transferred as an image or document for design and implementation purposes.
- *Screen prototype*. The prototype is built as real screens using a tool and shown to the User. However, UI elements do not respond to the events such as clicks.
- *Functional prototype*. The same as the screen prototype but some important UI elements such as submit buttons or drop-down lists function as expected on sample data.

The UI models as presented in this thesis are tool-drawn mock-ups. We draw the prototypes using different diagramming tools and use the generated images in this document. In practice, these are close to be screen prototypes. The User – i.e. the developer – builds them using a tool and transforms them to models readable by our utilities using automated tools. They could also be seen as functional because the UI models in this thesis are connected to behaviour and data elements that represent the functionality of the system at an abstract level.

2.8 Web Engineering

The engineering of web-based applications, services and information systems has evolved as a branch of software engineering. Web engineering is now widely accepted as an independently recognizable discipline. According to [Web2011], web engineering is “the application of systematic, disciplined and quantifiable approaches to the cost-effective development and evolution of high-quality solutions in the World Wide Web”. Development of web applications differs from conventional software development in many respects. According to Mendes and Mosley [Mendes2006], the main differences are about the people involved in the development of applications as well as those considered the application Users, and the inherent characteristics of web applications:

- Conventional applications are still being dominantly developed by IT professionals while web development requires people with a wider variety of skills and expertise such as interactive multimedia designers, content developers such as writers, and artists like photographers. Also, many non-experts are involved in the development of customizable web applications enabled by the

wide range of tools and packages. Examples are Facebook applications [Face2011], blogger widgets [Blog2011] and Flash Animations [Flash2011].

- Compared to conventional software applications, web applications are typically aimed at a larger and more diverse group of users. An even more important characteristic of the users of web applications is the fact that they are unknown for the most part. Some web applications tend to target more specific groups of users but nevertheless users can come from any background as long as the application remains publicly accessible.
- The types of technologies used for web applications make web application development specific. Web applications rely on hypertext and hypermedia scripts on the client side. These applications use communication technologies for database access while conventional software applications utilise more direct and local technologies to connect to databases. Web applications are developed knowing that they would be accessed from several platforms while conventional software are normally developed for the specific users' platform.
- Last but not least is the social impact of web applications that is missing in conventional software applications. Since the early days of web, there was a demand for more interactive web pages, where users can interact with each other and with the page administrators. The social force has caused many approaches to enable flexible user interactions in the past and hence Web 2.0 has been introduced. Web 2.0 is an indication of a social need rather than a technological shift but technologies have also played their part.

Several Web 2.0 technologies and platforms exist. For the most part, these deal with the client-side technologies; at the server-side, Web 2.0 applications often use the conventional web technologies. The most important technologies stemmed in two groups defined based on two frameworks: Ajax [Garr2007] and Flash [Flash2010]. While Flash-based technologies are more appropriate for applications with large data grids and heavy user interactions, Ajax results in more general web applications. HTML 5.0 [HTML2010] is another recent advancement in this area. HTML 5.0 performs almost as well as Flash⁹ on the client side with much less complexity.

⁹ Flash is being deprecated by Adobe.

Google Web Toolkit (GWT) [GWT2010] is an Ajax-based platform targeted in this thesis for model-driven development of Web 2.0 applications.

2.9 Summary

In this chapter, background concepts and terminology related to this thesis were discussed. Different approaches for the separation of concerns were discussed among which we specifically detailed MVC and MDA. We mentioned that MDA supports three levels of modeling that are CIM, PIM and PSM and that our abstract model is termed Abstract-Platform SM (APSM), for it is independent of specific web platforms and yet it is specific to the abstract web as a common platform. Then, we discussed the details and contents of the PIM, APSM and PSM and we reviewed different approaches for mapping between these levels. Since UI prototypes are included in our PIM, we introduced basic concepts of UI modeling and prototyping. Finally, we listed some differences between conventional software engineering and web applications engineering. It can be concluded that the ideas of this thesis belong to the area of web engineering or more specifically to the area of Model-Driven Web Engineering (MDWE). Appendix D contains a brief definition of the most important terms discussed in this chapter.

Chapter 3. LITERATURE REVIEW

In Chapter 2, we introduced the basic concepts relevant to this thesis. In this chapter, we review the existing model-driven web development approaches. First, we list the requirements for a web model in Section 3.1. Then, based on these requirements, we present the related work in four sections.

In Section 3.2, we discuss approaches that have not had direct impact on this thesis but present interesting ideas about the requirements in Section 3.1, and approaches that help obtain a general understanding of model-driven web development. Section 3.3 presents related work that has had a direct effect on the subject of this thesis. Two of those are particularly important and are discussed in Sections 3.4 and 3.5. In Section 3.4, we present the details of a web modeling language from which our approach for visual modeling of web applications was inspired. Section 3.5 elaborates on an abstract UML-based meta-model that is used in this thesis as basis for the APSM.

In the chapter conclusion laid out in Section 3.6, we present an analysis of the related work discussed in Sections 3.3, 3.4 and 3.5 in regard to the requirements listed in Section 3.1 and outline arguments for the need of a new abstract model for model-driven web development.

3.1 Requirements of an Abstract Web Model

In this thesis, we aim to design a web model with the most common features as well as the most required ones. An abstract web model should allow modeling of common properties and features present in the existing web models as well as web engineering coding frameworks. In this section, the existing web models and web frameworks are reviewed to infer a list of common features (requirements) for web development. Because of our focus on MDD, we are also interested in identifying requirements that facilitate the usage of MDD techniques.

Most web application frameworks are based on the separation-of-concerns principle using the Model-View-Controller (MVC) or a similar pattern. Examples of frameworks that promote such separation include ASP.NET [ASP2010], ColdFusion on Wheels [Cold2010], Apache Struts [Strut2010], Java Server Faces (JSF) [JSF2010], JBOSS Seams [Seam2010], Spring, Ruby on Rails [Ruby2010], Aida/Web [Aida2010] and Catalyst [Catal2010]. The most

recent version of the .Net framework – compared to earlier versions – has moved further on by providing default controllers and stipulating strict packaging rules for separation of concerns in accordance with MVC 2.0 [MVC2011]. The notion of “controller” is central in the MVC structuring scheme. Some frameworks such as Google Web Toolkit (GWT) do not explicitly support MVC, but supplies entry points and an asynchronous event-management mechanism for controlling the application. Based on the established practice, we can therefore infer that a web model needs a controlling feature composed of a controller class and its supporting operations. The controller acts as a central decision point for the application. The implementation of the controller may be according to the original MVC pattern [Schmid2000], variations of MVC such as Meliá et al.’s [Meliá2005-1] or non-MVC patterns such as the ones of service-oriented applications [WS2011]; this is left to the target platforms.

Most of the existing web application frameworks support data-related operations such as recording, retrieving, updating and searching through data in a database or a file. Some, such as Ruby on Rails, provide database migration functions. PHP [PHP2012] has functions specific to certain database technologies such as MySQL [MSQL2012]. A variety of approaches are used by different web application frameworks for data access, data interchange, data format conversion and data presentation. All these features need modeling and implementation of data entities. Most of the existing web modeling languages such as [Ross2006], [Botter2007] and [Baresi2006] also model data entities as more web applications become dependent upon data. It is therefore necessary to model abstract data entities and data handling in an abstract web model. The data supplied through the data interfaces of the existing frameworks is often required to be serialized – e.g. using XML or JSON [JSN2012] – to be transferable between clients and servers. Hence, there is a need for two types of data containers, one to be used at the data access layer and a second serializable one to be transmitted in between layers.

Security features have become an essential part of web applications. An abstract web model needs elements representing secure pages, operations and services as well as login/logout mechanisms and session/page variables, as most of the existing frameworks such as .Net, Ruby and Hibernate do. Some other frameworks such as CppCMS [CPPCMS2010], Fusebox [Fuse2010] and GWT do not supply such features as a part of their framework, yet these frameworks can also be used for implementing security features. For example, GWT is widely based on Ajax and Google services that require and enable secure connections.

A web application framework is not usable without a support for the creation of user interfaces (UIs). Most existing frameworks such as GWT, .Net, Struts and J2EE provide their own UI types and components. UI components trigger events and use data. Consequently, an abstract web model should supply different abstract UI components and support processes that relate them to operations required to get data and to trigger events.

Web 2.0 has facilitated the development of more interactive, searchable and social web applications. Several web application frameworks such as GWT, ColdFusion on Wheels and Apache Struts integrate support for Ajax [Garr2007], which is now seen as the most popular Web 2.0 technology. Other Web 2.0 technologies such as Flash [Flash2010] and HTML 5.0 [HTML2010] are also used, especially with regard to mass data processing and multi-media applications such as media sharing sites and mobile applications. Web 2.0 requires that separate parts of a web page be updatable independently, the contents be searchable and the user be able to interact with the contents in real-time [O'R2005]. In order to implement the framework must allow the client to invoke a server operation without having to wait for the results; that is an asynchronous call must be made to the server.

Since the solution of this thesis is MDD-based, it is important to consider requirements from MDD techniques as well. First, as established by Cicchetti and Di Ruscio [Cich2008], semantic relationships instead of name-based associations should be used in meta-models. MDD techniques often establish a mapping between elements of different viewpoints based on their names [Cich2008]. This is a useful technique but one that makes it difficult to trace back and to identify the origin of mappings because of changes applied by developers to element names as well as the one-to-many nature of most of the mappings. Another MDD-related requirement is the selection of an appropriate language for the meta-model. According to Brambilla et al. [Bramb2008], most of the existing web meta-models such as WebML and W2000 are broached as domain-specific languages (DSLs) with no roots to UML. A better option would be a UML-based model since a UML-based model can benefit from the popularity of an established standard as well as its integrability with other UML-based tools and methods.

Finally, we infer requirements by considering the software engineering point of view. First, lower-level models should be created in accordance with higher-level requirements [Chaos2001, Vern2005]. Although there is a limited capacity for directly addressing this aspect in an abstract web model, different researchers and practitioners [Molin2008, Escal2008,

Escal2003, Lee1998, Suh2001, Weid1998, Koch2006, and Lian2007] have observed that the more a web modeling approach includes the possibility to express requirements-related features, the more beneficial it becomes. Second, as with other technologies, a barrier that can prevent developers from using MDD techniques is their reluctance to learn a new complicated technique after becoming experts in one technology. To alleviate this problem, it is important that a web application can be defined using a simplified requirements definition language, and to provide automated transformations from that language.

In conclusion, the following are the main requirements for an abstract web model to be used for MDD:

- 1) Support for data access and event handling. This includes controlling mechanisms and support for data services.
- 2) Platform-Independence. This ensures the model is abstract and can be mapped to several web platforms.
- 3) Support for MDD necessities. This includes the fact that modeling elements, when required, must be connected using semantic associations. This category also addresses the ease-of-use by offering a simplified subset of the meta-model as an input language along with the required transformations to map the input to the complete model.
- 4) UML support. This is seen as a requirement for defining the meta-model to enhance the popularity of the meta-model and its portability with respect to other tools, models and techniques.
- 5) Support for Security.
- 6) Support for Web 2.0 in terms of dynamic UI components that can be updated without halting the entire web page.
- 7) Support for expressing requirements. This includes functional requirements as well as UI components that represent the expected UI prototype of the application.
- 8) Support for separation of concerns.
- 9) Web-Specific Features. An abstract web model needs to support web-specific features. This support is required in terms of:
 - a) Structures that are specific to Web such as Web forms and hyperlinks – some of such features are now introduced to non-web applications as well.

- b) UI elements that must be supported at an abstract level; that is an abstract web model may not detail many aspects such as color, size and position of web elements as these are nowadays specified using style sheets and other similar technologies. This also affects the adaptability to Web 2.0.
 - c) UI components such as buttons, inputs and tables must be supported.
 - d) UI components must be integrated with the web application so one can model the interactions of presentation and controller at an abstract level.
- 10) The details of the abstract web model must be available so that different platforms and mappings can be precisely defined and supported. This is, albeit, not a quality criterion for rating an abstract model but the one that makes an abstract model extendable and portable.
- In Section 3.6, we will return to this list and will compare the existing approaches using these items.

3.2 General MDWE-Related Approaches

General related approaches share aspects with this thesis. Although these approaches have not had a direct influence on this thesis, studying them have helped us develop our own ideas of model-driven web development and hence indirectly helped improve the quality of this thesis. Section 3.2.1 discusses related work that share ideas with our work. Other generally related approaches are discussed in Section 3.2.2; these are still related to MDWE but have ideas different than ours.

3.2.1 Related Approaches

A group of the existing abstract models address web applications but do not specifically describe the details of UI modeling. These models are primarily concerned with functional aspects. Nikolaidou and Anagnostopoulos [Nikol2005] suggest a common UML meta-model for web-based information systems to tackle performance problems in the functional and physical layers. The importance of Nikolaidou and Anagnostopoulos's approach is their effort to address abstract data handling operations efficiently. W2000 is a meta-model for web applications that focuses on performance [Baresi2006]. W2000 provides a multi-layer architecture for web-based applications in accordance with MVC. In another note, He et al. [He2005] report on an abstract method to define a role-based abstract model for web-based applications; He et al.'s approach is

based on recognizing functions and processes of all roles. He et al.'s approach avoids dependence to specific platforms; an objective shared by this thesis.

Another group of the existing models address abstract UI modeling in general. Blankelhorn [Blank2004-1] has provided a UML profile for GUI layouts. Our approach is related to Blankelhorn's as we also cover abstract UI modeling using UML. Da Silva and Paton [DaSil2003] describe a how-to for designing user interfaces using UMLi, an extension to UML, and compare the results with UI models built using UML itself. UMLi is related to our approach as both require a UML-based abstract UI model. Vanderdonckt [Vander2005] presents an MDA-compliant environment for designing user interface models of information systems in general. Vanderdonckt's approach is relevant as it is a step toward combining UI modeling and MDD. Schattkowsky and Lohmann [Schat2005] introduce a general method for designing platform-independent UIs. Schattkowsky and Lohmann's approach is related to our approach as it is platform-independent and supports UI modeling, which may be applied to any specific meta-model. Diamodl [Diam2008] is a modeling language to describe the data-related logic of UIs. This includes links, data flows and data collection gates. Diamodl is very limited and cannot scale up to cover higher-level aspects of UI modeling because it only models the data interchange between abstract atomic data units at the programming level. But the idea of modeling the data-related logic at the UI level using a graphical illustration is helpful for the design of visual web modeling languages such as the one in Section 5.1.

3.2.2 Other Approaches

Various efforts were devoted to synthesis of UI models. Li et al [Li2000] illustrate a how-to UML-based approach to devise the architecture of web-based applications in a three-tier framework. Kavalajian [Kaval2007] and Bogdan et al. [Bogdan2008] describe a method to extract a UI model from a discourse model in a general sense. Costa et al. [Cost2006] present an MDA-compliant method to devise a step in user interface design for web applications in accordance with the UML version of ConcurTaskTree (CTT) based on *Canonical Abstract Prototype* (CAP); an abstract language for the definition of user interfaces. The goal of the method is to transform the User Interface Definition (UID) model to a UML-compliant specific model for interoperability. The PIM in this method is use case based. Both the PIM and PSM include a Conceptual Architecture, Presentation Model and Dialog. The authors present a sample mapping from CAP to HTML (used as a specific platform). The transformation from PIM to

PSM and then to code is performed using the PHP application *Model2Code*. Another model-driven approach to generate web UIs was proposed by De Souza et al. [DeSouz2008]. The method covers the layers of MDA as follows: CIM with use cases, PIM with analysis and presentation models, PSM with design, navigation and UI design model. The method starts with use cases and spans through analysis, presentation, design and navigation models. The final result is a UI design model, which is followed by the implementation of the UI. An implemented Eclipse plug-in hard-codes transformation rules required to generate JSF code. Sukaviriya et al. [Suka2007] introduce a process framework for model-driven UI design based on iterative and incremental approaches. Nunes and Shwabe [Nune2006] present a combined approach of model-driven development and domain specific language for rapid software prototyping of web applications. Based on this approach, they have designed a tool named *HyperDe* that takes conceptual instances, a navigation model and an abstract UI, and then produces HTML pages.

Other related works include approaches that enable developing parts of web applications. As an example, Freudenstein et al. [Freud2008] present a modeling language to specify web dialogs. The language includes a domain specific language (DSL) based on Petri-nets and a Domain-Interaction Model. The latter is an interface that connects the models to the DSL. The interface works with the application that provides the model *Solution Building Block (SBB)*. SBB is a software component that Freudenstein et al. have developed to execute the dialog model. The modeling notation allows defining data elements and their interactions as well as user interface components used within the dialogs. Some approaches such as the Logic-Based Hypertext (LHM) introduced by Wan et al. [Wan1995] and the General Hypertext Data Model (GHDM) by Wan and Bieber [Wan1996] are early efforts to integrate classic information systems with hypertext features. Another similar approach is taken by Whitehead et al. [Whit2004] to bring repositories to the hypertext level so that they could be browsed on the web.

Several papers have been published on supporting architectural aspects of web applications in a model-driven sense. A general discussion on the basis and necessities of such architecture is provided by Taleb et al. [Taleb2007]. Hammoudi et al. [Hamm2008] 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. [Li2000].

Usability issues of MDD tools have also been addressed. Web-Site Design Model (WSDM) [DeTroy1998] suggests defining a web system based on its users' groups. A wizard-based approach to derive the user-interface model required to support existing domain models is suggested by Stocq and Vanderdonckt [Stocq2004]. A technique for classifying distributed user interfaces is reported by Demeure [Demeur2005] using a reference model. Sousa et al.'s work [Sous2007] is an effort to resolve usability problems of the tools and techniques used for model-driven user-interface development by employing a strategy that aims at usability issues as the top-most priority.

There are other model-driven approaches to web engineering that provide supporting solutions using MDD. For example, a model-driven methodology for composing web applications using model-driven techniques was published by Kateros et al. [Kater2008]. The method uses model-driven references and transformations to configure web applications. Vara et al. [Vara2007] provide a model-driven method for the generation of object-oriented databases to be used in a web information systems method named MIDAS [Cácer2003]. A formal approach to support web information systems with multiple data sources is presented by Vdovjak and Houben [Vdov2005].

An independent track of research is devoted to re-engineering or re-use of legacy web-based systems. An approach to reverse engineer UML class models from web pages is provided by Pu et al. [Pu2008]. Another approach for tracking customized manual code added to automatically generated applications is provided by Cichetti et al. [Cich2007]. Nguyen and Chun [Nguy2006] present an effort to synchronize interaction diagrams representing use cases with legacy code using model-driven techniques.

Modeling requirements has inevitably become a step in model-driven web development. Scenario-based Object-oriented Hypermedia Design Methodology (SOHDM) ([Lee1998] and [Suh2001]) is one of the first methods proposed to deal with web requirements. SOHDM uses scenarios as a technique to elicit requirements [Weid1998]. An approach addressing requirements models of web information systems is published by Molina et al. [Molin2008]. Cuaresma and Aragón's method [Escal2008] is a navigational-based method to model requirements of a web-based system. This requirement model is then used in a model-driven context. The method is supported by a tool named NDT [Escal2003]. An approach that generates a full web application from requirements is suggested by Liang et al. [Lian2007]. This approach

re-uses two different toolkits for requirements definition in natural language and application generation however the main contribution of this approach is a method for converting the models from the former toolkit to the latter one. Uden et al. [Uden2008] present an approach based on Activity Theory [Enges1987] for eliciting organizational and navigational requirements of web applications.

Guidelines, frameworks and principles 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 uses NDT tool is Escalona et al.'s approach [Escal2009]. The NDT suite provides a set of tools 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 for all model-driven development methods. Such a conceptual framework for MDA-based software development environments is described by Pastor and Molina [Past2007]. Since an ultimate goal of our approach could be the development of an MDA-based environment, Pastor and Molina's work provides a useful point of reference.

Finally, Web 2.0 has also been addressed in the model-driven community. For example, Valverde and Pastor [Valv2009] describe a meta-model to specify technical details of a Web 2.0 platform based on OOWS [Fons2003]. These include the features related to Rich Internet Applications (RIA) such as UI widgets as well as the event handling mechanism.

3.3 Related Approaches

In this section, we elaborate on related approaches that have had impacts on the approach discussed in this thesis.

3.3.1 Unified Web Engineering (UWE)

The series of work based on UWE [Kroiß2008] provide a language and a method for model-driven web-based development. The core UWE includes the following packages:

- *Requirements*, used to define the requirements of a web-based application. A method is presented for the definition of requirements based on use case and activity models. This is done with the help of new concepts that are specific to web applications such as a *Navigation* use case or *Browse* activity.

- *Navigation*, which defines different paths through the elements and hierarchical structure.
- *Process*, which provides methods one must follow to generate the application.
- *Presentation*, containing both the structure and elements used to communicate with the User. This is performed with the help of UI sketches that generate a type of UI model.
- *Content*, which is what the application presents as the content of the visible components to the User as well as the way this information is organized, managed and related to the database.

The UWE process is an extension of the Rational Unified Process (RUP) [RUP2009]. The authors extend the Inception, Elaboration, Construction, Transition and Maintenance phases of RUP by providing web-specific components that steer the process. A precise definition of the system is provided using a specific constraint language. The method follows these steps:

- requirements analysis
- conceptual design
- navigation design
- presentation design

Formal transformations are 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. The papers and documents provide general guidelines about the transformation process but the automated rules and method are partly reported for a specific *Spring* framework [Sprin2008]. The existing implemented transformation uses the ATLAS environment and samples of ATL transformations [ATL2005] are provided. The method covers all the layers of a web-based application but it does not generate data related operations required to fetch or store data. Koch [Koch2006] describes the transformation mechanism of UWE.

UWE bears its own abstract model of web-based applications as well as an abstract UI model. The flow of presentation is modeled using *Window*, *Frameset* and *Frame* classes, where *Window* is an aggregation of *Framesets* and *Frameset* is an aggregation of *Frame*. This basically means that windows are main units of presentation. Framesets are used to group different Frames within the same Window. Actual input/output forms are equivalent to Frames. This structure is recursive to allow for more complex paths. The UI presentation is modeled using classes

UIView, *Presentation* and *UI Element*, where *UI View* is an aggregation of *Presentation* classes and a *Presentation* class aggregates *UI Elements*. *UI Element* is an abstract class, which could be realized using actual presentation components such as text, buttons and menus.

UWE provides detailed guidelines along with informal rules for creating UI elements and navigation models from analysis models. A requirements modeling step is also described [Koch2006]. The essence of the UWE is manual even with regard to data and presentation design unless applied to other MDD approaches with automated steps. The reason is that UWE as its origin – RUP – acts as a framework rather than a concrete method. A more proper usage of UWE, therefore, is to define other MDD – such as the one in this thesis – following the UWE framework.

3.3.2 Object Oriented Hypermedia Design Method (OOHDM)

The Object-Oriented Hypermedia Design Method (OOHDM) is a methodological framework based on the Object-Oriented Hypermedia (OOH) [Gomez2000] model. An approach based on OOHDM is the one by Rossi and Schwabe [Ross2006]. This framework is a data centric object-oriented model for hypertext meta-modeling. The framework proposes the following model-driven steps to generate a web application:

- Requirements Engineering
- Conceptual Modeling
- Navigation Design
- Abstract Interface Design
- Implementation

The requirements engineering step is based on use cases. The use cases are claimed to be supported by interaction diagrams but what the authors present as interactions diagrams are very similar to state machines. The reason is that those diagrams depict the transitions from one page to another one as interactions that happen between units. An abstract UI model is provided that needs to be extended for each application. There is also a mapping from the OOHDM to Java 2 components in accordance with the MVC framework. Since the developer uses Java mappings and generates Java applications, the approach becomes platform-specific although the model is abstract.

The conceptual design is realized using class diagrams while several models are used for navigation design amongst which state charts are the core element. The abstract interface design is performed in terms of objects, their interfaces, events and response objects provided in the context of an interaction diagram. No explicit model is suggested for the implementation of the approach.

Other approaches based on the OOHDM have been proposed. In particular, Schmid and Donnerhak's approach [Schmid2005] distinguishes two PSMs: conceptual and navigational. An advantage is the possibility of the choice of independent technologies for the presentation and logic. This complies with the notion of platform in model-driven architecture. However, the approach only covers the navigational transformations. Navigation is separated into paths to fixed and dynamic pages; navigation to dynamic pages requires an operation to build the contents according to given parameters. A semi-formal language is used to generate the PSM. Another model-driven tool-set based on OOHDM is OOH4RIA [OOH4RIA2011] that generates applications using GWT.

3.3.3 Wu et al.'s approach

Wu et al. [Wu2007] describe an approach to generate user interface code based on MDA transformations and the Model View Controller (MVC) pattern. The approach spans the gap from requirements to code for a user interface model by transforming boundary objects resulting from a robustness analysis [Rosen2007] to JSP pages [JSP2008]. In order to do this, Wu et al. provide a process 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 according to transformation rules and UML models. The process is as follows:

- Requirements are defined as use cases. Activity diagrams describe the use case behaviours; this forms the CIM.
- The CIM is then mapped to the PIM based on robustness analysis, at which time control, boundary and entity objects are highlighted
- The PIM is automatically transformed to a PSM consisting of Java Widget class diagrams representing the View
- The authors provide informal descriptions to generate other parts of the PSM including a class diagram as the Controller and an ER schema as the Model

- The code includes JSP pages automatically created from Widget diagrams that model the rich user interface.

3.3.4 Web Software Architecture (WebSA)

The Web Software Architecture (WebSA) is an OOH-based approach that aims at covering all phases of Web application development. WebSA is an architecture-centric approach. It uses Java Metadata Interface (JMI) [JMI2008] as a way of defining meta-models for the implementation and uses them as basis for other models. The approach uses WebTE [WebTE2008] as a transformation engine. However, it does not provide specific transformations; instead the application designer must supply the transformations for every application.

WebSA provides default transformations that apply to a subsystem and configuration viewpoints, and results in an architectural viewpoint. This latter viewpoint is used along with a functional viewpoint for mapping to actual applications using a set of customized transformations. WebSA generates models for the presentation, the core functionality and content but do not cover data access mechanisms. The transformations are based on the mappings between different architectural viewpoints but do not map to implementation-level models of web applications. The WebSA process is summarized as follows:

1. Developer generates models according to WebSA framework and the functional web modeling of OO-H.
2. WebTE transforms the generated models to a JMI-based model
3. The developer defines the transformations to map high-level models to detailed architecture models using the UML Profile for Transformations (UPT) [Meliá2006]
4. The developer can choose to see the transformation compiled as Java code
5. The models are mapped to an integration model, which integrates the functional and subsystem viewpoints
6. Finally, the integrated model is mapped to an implementation model, which can be executed.

As one can see, the transformation must be defined by the developer. Notice that the WebTE is a transformation engine but not a full-fledge CASE tool.

3.3.5 Ubiquitous Web Application (UWA)

The Ubiquitous Web Application (UWA) standard is an effort by the W3C to standardize the definition and transfer of web-based applications. The goal of UWA is to facilitate the access to web applications to users with different goals. Our idea of abstract web is close to UWA but UWA models are independent from implementation details. A UWA-based model is not intended to specify features related to web-based applications technology such as the architecture, the software or the database.

Distante et al. [Dist2007] present a model-driven approach that uses UWA along with MVC to generate web-based applications using Java Server Faces (JSF) [JSF2010]. The authors provide a method to map UWA models to platform-specific models. These platform-specific models are bound to the presentation layer as UWA mandates. The approach presents a general UWA-based method and maps it to an abstract model-driven method. Distante et al.'s approach is tightly MVC-based as it classifies main modeling elements into two sets:

- Model Class Diagrams that contains data objects
- View Class Diagrams that include an object-oriented view of the presentation classes representing pages, links and the UI components.

The controller operations required to control both types of models are also defined.

The general method and models are platform-independent but customized transformation rules that map the high-level model to specific platforms are needed. The process is as follows:

1. Define a CIM in terms of UWA requirements elicitation standards.
2. Map the CIM to an UWA conceptual model, which is composed of *Information*, *Navigation*, *Transaction/Operation*, *Publishing* and *Customization* submodels.
3. The above models are once again transformed into an integrated logical model at the PIM level.
4. The logical model is mapped to the JSF platform. This is done by synthesizing a model consisting in a Model, View and Controller sub-models and merging these sub-models according to the mapping rules.

3.3.6 Muller et al's approach

Muller et al. [Mull2005] describe an approach for the development of web-based applications. The method divides platform-specific models into two groups:

- Platform-dependent PSMs, which are dependent to the general concept of a platform without explicitly mentioning the specific technologies used to build the application. This is essentially equivalent to our notion of abstract web.
- Technology-dependent PSMs that are the mappings of platform-dependent PSMs in terms of specific technologies used to implement those models.

A web application is modeled in three layers:

- Business Model: essentially the domain model composed of objects that are important from the viewpoint of the business model.
- Hypertext Model: the organization of web pages and the navigation path throughout the whole application.
- Presentation Model: the model of what the User can see and communicate with.

The approach is supported by a tool called *Netsilon* [Mull2005]. Netsilon supports a special action language called *Xion* [Mull2005]. Xion is used to model the behavior at the hypertext level and is based on OCL. This language is used to model the behavior of operations at an abstract level. This is also accompanied by an abstract SQL syntax used to model data access services. The method supports transformations from these data access services to some relational database management systems.

A modeling profile is also provided to model web applications. This model is composed of:

- Web Files and Zone, which define the general organization and web pages.
- Decision Centers that capture the relationship of web pages. For example, a form containing a submit button is such an element. Clicking the submit button causes a transition from one page to another one.

The automated transformation happens at the last step, where the above mentioned platform-dependent PSM is transformed to an executable implementation of technology-dependent PSMs. The formalism for defining the transformation is not specified. The approach

provides models and tools for other transformations such as PIM-to-PSM but not as a formal implementation.

As mentioned, Muller et al. break up the PSM level into two levels: One is named platform-dependent PSM and the other is technology-dependent PSM. Such a separation results in transformations from PIM to platform-dependent PSM relatively longer than the ones from platform-dependent PSM to technology-dependent PSM. A difficulty with this approach is thus to identify and differentiate the terms platform and technology.

3.4 The Web Modeling Language (WebML)

The Web Modeling Language (WebML) is introduced as a modeling language for designing web-based applications by Ceri et al. [Ceri2000]. This language provides a notation to define the UI presentation layer and the semantics to specify web-based applications in a multi-layer approach. According to WebML, a web-based application is defined in terms of: *Data model*, *Hypertext model*, *Personalization* to give different users different viewpoints, *Presentation* to add the look-and-feel, and *Integration* of business processes and Web services.

WebML provides a notation to define web-pages and processes as well as the operations and services that are invoked from web pages. It also supports data modeling and data access mechanisms. A WebML-based approach would support the design and development of data-centric approaches for web development over the following layers:

- Data model
- Hypertext model
- Personalization to give different Users different viewpoints
- Presentation to add the look-and-feel
- Business Processes Integration Layer
- Web Services Integration Layer

WebML includes an internal XML-based mechanism for model management. Although the approach consists of several phases, it is mainly concerned with data and hypertext modeling through which the method generates the executable code. There is no explicit separation between

PIM, PSM and the implementation. It is possible to obtain the generated code but the PSM is not available as a separate artefact. Transformations are stored in an internal XSL-based format.

In term of data modeling, WebML proposes the use of entity-relationships diagrams or an equivalent subset of UML. It is possible to use transformations that infer *derived* attributes calculated from other attributes. Data units can be of either the following types:

- Data units: basic data instances
- Multidata units: a set of data instances
- Index units: key attributes to describe instances when it comes to selection
- Scroller units: enables browsing a set of data instances
- Entry units: that collect input from users

In term of hypertext design, WebML supports a hierarchy of *Site View*, *Area*, and *Sub-Areas* or *Pages* recursively. Modeling units are either *content* units that are devoted to contain hypertext content or *operation* units that facilitate the controlling operation and data-related operations. Pages are categorized as:

- Homepage: the default page of a website
- Default Page: the default page of each area of a website
- Landmark Page: pages that are accessible from every other page such quick links
- Other Pages: pages that are neither of the above.

WebML was introduced before the advent of MDD/MDA. An effort was made subsequently to adapt to MDD/MDA [Bramb2008]. WebML has also been extended to support web-service development, business process and workflow modeling. For example, Tongrungrrojana and Lowe [Tong2004] use WebML in order to present the Web Information Exchange Diagram (WIED) at a higher-level of abstraction. Their objective is to address the inefficiency of the existing web/information modeling languages when dealing with information exchange models at the level of business processes. This is a companion to Tongrungrrojana and Lowe's previous work about WebML+ [Tong2003], a language for modeling information exchange in the workflow models.

Approaches based on WebML rely on WebRatio [WebRat2008] for web-application development. WebRatio supports a manual process for model development but executable code is automatically generated from the WebML model, as a Java application running on an Apache Tomcat [Apach2012] server. WebRatio also supports requirements analysis according to the following steps:

- Identification of group of Users
- Specification of functional requirements for each group of Users
- Identification of core information objects
- Decomposition of web application into web sites and their internal structures
- Conceptual Modeling of the hyper-content
- Data design with regard to data objects and sets
- Hypertext design that goes on top of data modeling and defines different links, parameters and operations

WebRatio [WebRat2008] presents a comprehensive model-driven web development approach and generates fully executable code. However:

- WebRatio enables script units where Ajax code for Web 2.0 support can be injected. However, WebML does not support Web 2.0 directly as it lacks the structures and processes required by Web 2.0 as outlined by Wallis et al. [Wall2011]. One needs to manually create the required Web 2.0 features for every component of the web application.
- The code generated by WebRatio is platform-independent in the sense that it is Java code that could be deployed on virtually any web server. But in practice, platform-independence cannot be achieved just by having Java code.
- WebML is not based on either UML or MOF according to Brambilla et al. [Bramb2008]. Although there have been a number of efforts to adapt WebML to MDA/MOF/UML such as Moreno et al. [Moren2006, Moren2007], Schauerhuber et al. [Schauer2007] and Brambilla et al. [Bramb2008], the tools and techniques based on these models are still using the original DSL versions. More specifically, it is not possible to map WebRatio models with UML-based tools [WebRat2011].

- WebML is specifically designed for building web applications – as it is a DSL.

WebML has been successful in proposing a visual language that developers can use to define applications. Our visual language for the definition of input models used as starting point for model-driven transformations was inspired from WebML.

3.5 The Botterweck Model

Botterweck [Botter2007] presents an abstract model for specifying web-based applications supported by a UI model and a data model along with connections to UML state machines. Botterweck's model aims at specifying multiple UIs. It extends the UML in order to allow modeling of applications in different layers based on MVC. Botterweck's model includes the following packages:

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

Figure 7 details the UI Structure package. According to this package, an application can have several user interfaces. A user interface is a special kind of UI composite, where a UI composite is a collection of UI components. UI components are elements that the user can see and directly communicate with. A UI composite may be a presentation, which in association with a state makes a presentation state meaningful. This package also contains the abstract elements used by concrete UI elements in a UI element model.

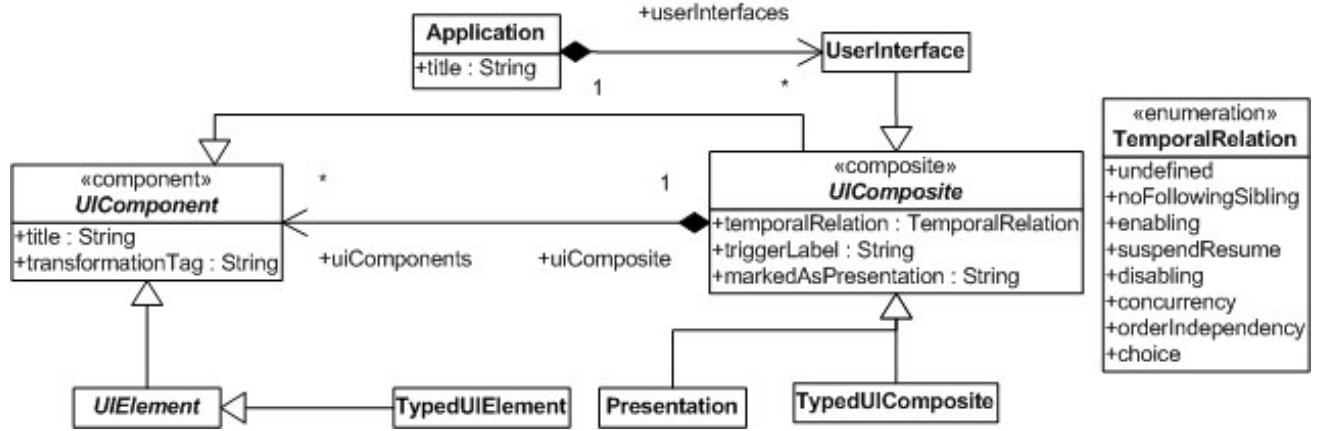


Figure 7 The UI Structure Model from [Botter2007]

Figure 8 shows a subsection of the UI components model. These components include: components connected to data elements, and composite elements such as input fields, selection boxes and table views. Figure 9 presents the model for plain hypertext, links and buttons. Triggers occurring in a presentation can be of one of the following three types: hyperlinks, operation triggers such as submit and navigation triggers that only cause a change in navigation.

Some other characteristics of the Botterweck model relevant to our research are that a transition can be associated to several operations and events (Figure 10). Data Oriented elements are associated with Data Composites (collections of data elements). Data composites are associated to a special class called *OperationAdapter*. One of the following roles (selectOperation, insertOperation, deleteOperation, updateOperation) corresponding to CRUD operations can be used for the association (Figure 11).

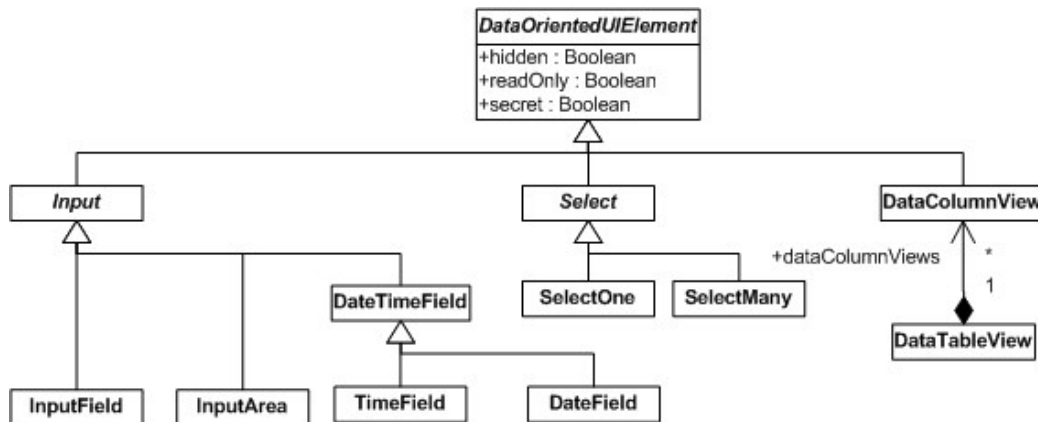


Figure 8 Data-oriented elements of UI components from [Botter2007]

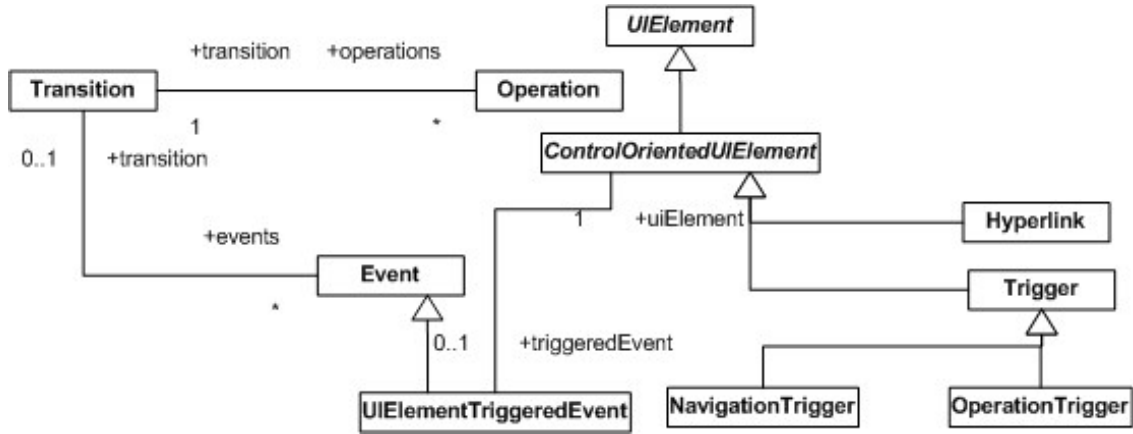


Figure 9 Control-oriented elements of UI components from [Botter2007]

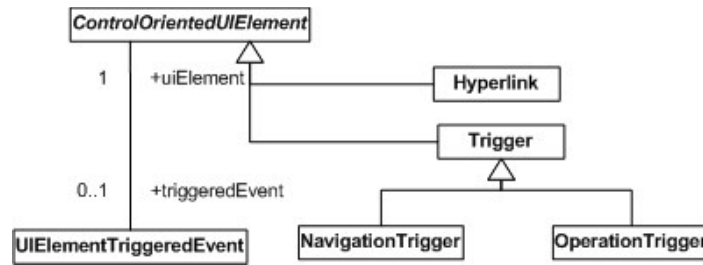


Figure 10 Transition in relation with events and operations from [Botter2007]

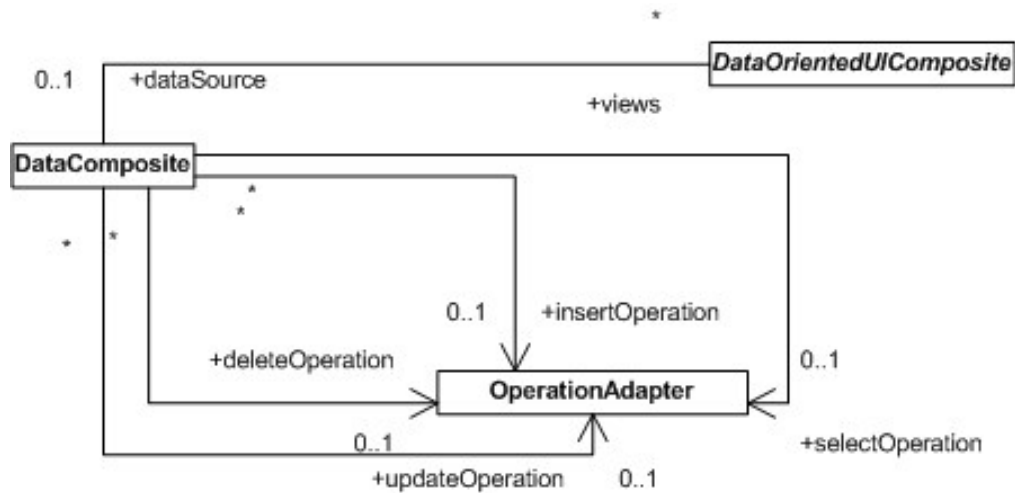


Figure 11 Data Composite related to Operation Adapters [Botter2007]

3.6 Comparative Study of the Existing Approaches

Table 2 lists some criteria used to compare the existing models in accordance to our requirements listed in Section 3.1:

- **Data Access Operations:** indicates if the meta-model supports operations for data handling at an abstract level. (Item 1), Section 3.1)
- **Platform-Independence:** refers to the fact that a model is abstract and also that the model has been validated against at least two different platforms. (Item 1), Section 3.1)
- **Non Name-based Dependencies:** means that the model does not depend on the name-based analysis rules to interrelate elements of different viewpoints. (Item 3), Section 3.1)
- **UML-Compliant:** refers to the fact that the model extends UML, although it is not necessarily expected that the model follows the formal extension mechanisms. (Item 4), Section 3.1)
- **Web-Specific Features:** means the model supports web-specific features in opposition to the abstract-web features. (Item 9), Section 3.1)
- **Web 2.0:** indicates if the model explicitly supports Web 2.0. For example, in WebML, one may use script units for inserting Ajax code to support some Web 2.0 capabilities but this is only a work-around. We rather look into a model that allows every element to be searchable, to accept feedback and to be updatable independently and inherently. (Item 6), Section 3.1)
- **Requirements as Input:** indicates if the model integrates requirements with lower-level models. (Item 7), Section 3.1)
- **Abstract UI Model:** suggests that the model presents the elements required to model an abstract UI model. (Item 9), Section 3.1)
- **Availability of Details:** verifies if the details of the meta-model are available. With regard to MDD techniques, such details are necessary for extending the model as well as creating the transformations that map the model to other models, languages and tools. (Item 10), Section 3.1)
- **Coverage of UI Components:** points to the fact that a meta-model has elements for modeling specific UI components such as tables, buttons and input controls. (Item 9), Section 3.1)
- **Integration of UI Components and the Web Model:** means if supported UI components are integrated with behavioural and navigational aspects of the meta-model using meaningful associations. (Item 9), Section 3.1)

- **Separation of Concerns:** if the model addresses different viewpoints and suggests their proper integration. (Item 7), Section 3.1)

Table 2. Comparative Study of Existing Abstract Models¹⁰

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

As Table 2 shows, Botterweck's model scores higher in terms of support of the above elements. It is worth noting that Botterweck's model has a general UI model not restricted to web applications. An advantage of this model compared to web modeling languages such as WebML is that it is based on the UML - although recent efforts have focused on complying WebML with the UML [Bramb2008]. It is also worth mentioning that Botterweck's model provides support for advanced UI elements such as multimedia contents unlike most other models. Table 2 also draws attention to the fact that none of the studied models are grounded based on the requirements as an input along with models of UI, data and expected operations that satisfy those requirements. We selected Botterweck's model as basis for our model but as indicated in Table 2, Botterweck's model needed to be extended in order to address rows 3, 5, 6 and 7. This extension results in a new abstract web model, which is needed for the following reasons.

¹⁰ A star in a cell means the feature (row) is supported by the model (column) to the required extent. Lack of a star means inadequate support.

1. The existing models including the ones addressing data-centric or information-based web applications - such as the ones suggested by Ceri et al. [Ceri2006], Nikolaidou and Anagnostopoulos [Nikol2005] and Baresi et al. [Baresi2006] do not tend to cover the data access layer; rather they focus on the presentation and navigation required to present this data and information at the hypermedia level; WebML is an exception.
2. Most of the existing models are usually defined toward a PSM. Exceptions such as UWA [UWA2002] and Sakowicz et al. [Sack2007] end up with models that are very general and too abstract. Some, e.g. [OOH4RIA2011] and [WebRat2008], claim to be able to map to multiple platforms but the only indication to that effect is the fact that the generated code is Java-based. However, one may note that Java is only theoretically deployable on any platform. In reality, specific web platforms are made of several different technologies and customized with a variety of configuration settings. Therefore, the only way to prove a model could be mapped to more than one platform is by explicitly transforming the web model to the one of the platform and not just the Java classes; this is missing in the existing work. As a result, most of the research leads to approaches that are either hard to map to specific platforms or hard to adapt to other platforms.
3. As described by Cicchetti and Di Ruscio [Cich2008], most of the existing models are affected by the name-based mapping rules that prevent effective model-driven transformations be developed.
4. Most of the existing Web modeling languages, including WebML, are not based on UML/MOF according to Brambilla et al. [Bramb2008]. There have been a number of efforts to adapt WebML with MDA/MOF/UML family by Brambilla et al. [Bramb2008], Moreno et al. [Moren2006, Moren2007] and Schauerhuber et al. [Schauer2007]. These suggest that it is an advantage to design a model that is UML-based.
5. Only some of the existing models such as those found in [Hern2010], [OOH4RIA2011] and [Valv2009, Fons2003] provide partial or platform-specific support to Web 2.0. Some tools such as WebRatio provide script units, where the

developer can insert Ajax code for building Ajax-based Web 2.0 functionality but this is only a work-around at the tool level.

6. Finally, despite several approaches to incorporating requirements in the web engineering process [Molin2008, Escal2008, Escal2003, Lee1998, Suh2001, Weid1998, Koch2006, and Lian2007], the existing models still show little integrability with higher-level requirements as they do not provide the required mappings and semantics to connect the two.

The APSM and its supporting tools support all the requirements in Section 3.1. We will verify this in Chapter 9, after the presentation of the model and examples.

3.7 Summary

An abstract web model must fulfill different requirements regarding controlling mechanisms, UI components, data services, Web 2.0, security and the separation of concerns. Several model-driven approaches to web development exist, among which some provide abstract web models. WebML is one of the most popular approaches, which has been used in many tools, methods and studies. WebML is a DSL supported by the WebRatio tool. UWE is another general model-driven web development process. Muller et al.'s approach is close to our work with regard to the separation of PSM into two PSMs, technology-dependent and platform-dependent. Botterweck's model is selected as the basis of the APSM in this thesis because it covers most of the requirements of an abstract web model. The APSM would go beyond the existing work by its coverage to data-related features, well-defined model-driven associations and the integration of Web 2.0 necessities. It is also supported by platform-specific transformations that map the APSM to more than one customized specific platforms. Chapter 4 details the APSM and discusses its mechanisms to extend the existing work and to support the requirements in Section 3.1.

Chapter 4. THE APSM

In response to the requirements listed in Section 3.1, we designed an Abstract-Platform Specific Model of web applications (APSM) and present it in this chapter. The APSM is a meta-model for defining abstract web applications and for specifying transformations. The APSM extends the Botterweck model [Botter2007]. The structure and components of the UI model as well as the relationship between the UI model and data from the Botterweck model have been retained in the APSM. The parts related to behaviour have been customized. Modeling elements and associations are also added to support items 3, 5, 6 and 7 of Table 2. A summary of the list of our enhancements to the Botterweck model is presented in Section 4.11. Details of modeling elements of the APSM are documented in Appendix A.

4.1 Basic Terminology

We start by defining some of the web-specific terminology used throughout this chapter. Table 3 contains a list of specific terms that are repeatedly cited in this chapter.

Table 3.a. Web-Specific Terminology of Chapter 4	
Concept	Description
Presentation unit	An independent unit that displays a block of information such as a document, page, table, section or form.
Call event	An event that necessitates a call to an operation.
Operation trigger	A web page element such as a submit button, a list box or an active image that once activated triggers a call to a client or server-side operation.
Signal Event	Any event of the page that requires attention and will be directed to a client or server-side operation.
Front-end	A part of the web application that operates on the User's computer, this includes the downloaded contents, applied styles, and client-side operations.

Table 3.b. Web-Specific Terminology of Chapter 4

Concept	Description
Server-side tasks	Web application tasks that must be completed on server side such as database access, service invocation, session management or file sharing.
Client-side tasks	Web application tasks that can be completed on client side such as updating a field, reordering elements or validating user input.
Session Variable	A variable that holds some information about a specific time period when client and server are communicating. Session variables have a name, value and expiration time. Examples are session variables to hold a username, number of user attempts or last four digits of a credit card number.
Back-end	A part of the web application that operates on a server, e.g. application, web, data or email server.

4.2 A High-Level View

Figure 12 shows a high-level conceptual view of the APSM.

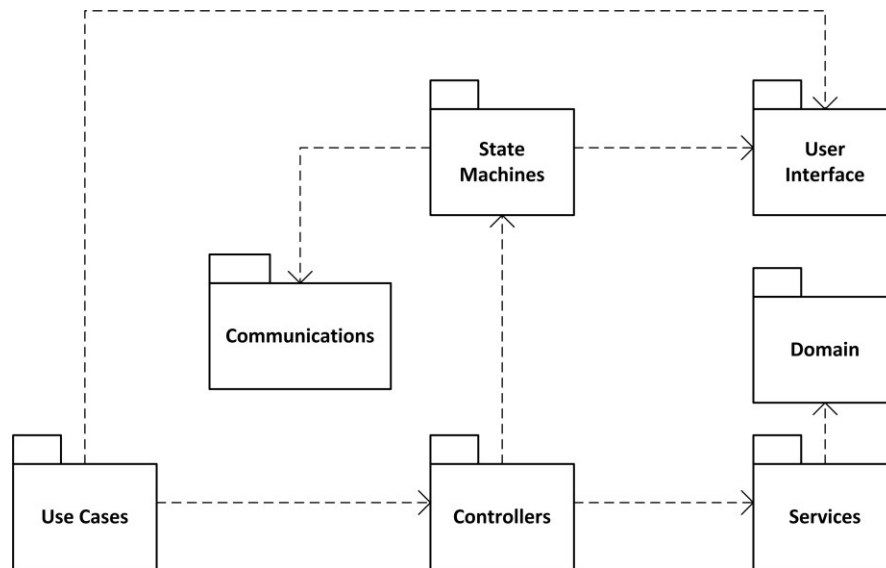


Figure 12 High-Level Architecture of the APSM

Elements of packages *State Machines*, *Communications* and *Use Cases* are mostly inherited from UML. The package *Domain* contains metaclasses required to define data entities, data composites and their attributes. The package *Service* contains metaclasses required to build data-access operations; these are dependent upon the *Domain* package to perform operations.

expresses an inclusion or extension relationship. The type of relationship depends on the state being preceded by a choice. The different elements of state machines are used as follows in the APSM:

- States are used to represent the status of a web application. A state shows a presentation unit or is used to specify information processing. The former is used at the PIM level as well. The latter is only used at the APSM level, where a state is used for the initiation of call events. States are also used for handling errors.
- Choices are used for building conditional flow of events especially for representing alternative use case steps and use case extensions. Outgoing transitions of choices are used for handling Boolean results of call events.
- Transitions can be associated with operation triggers, hyperlinks or use case inclusion/extension. Transitions can also have events and conditions. A transition-carrying event is used to transfer a presentation unit to a processing state and transitions carrying conditions are employed to show exit values of choice vertices.
- Events are assigned to transitions. We distinguish *call events* from *signal events*. Call events are used to represent operation calls to logical or data operations performed at the server or client side. Signal events model operation triggers such as submit buttons on a web form. Signal events carry the value of UI fields contained within the same form, where they are placed.
- Sub-state machines are used to specify use case inclusion or extension. A sub-state machine *ss* of a state *s* expresses that a use case corresponding to *ss* is included in the use case corresponding to *s* at the point of occurrence of *s*. If *s* is preceded by a choice, then the use case corresponding to *s* extends the use case corresponding to *ss*.
- Initial states are used as a formal start point as in UML with no further extension.
- Final states may be related to other use cases to show a sequencing relationship to another use case (state machine). Such a relationship means that the specified use case runs after the successful execution of the current use case.

The APSM extends the UML specification by allowing use case associated with a state machines state. Additionally, for navigation, such as a situation where a successful completion of

a task leads to another web page, it is necessary to indicate that a given use case sequentially follows another use case¹². This sequencing of use cases is indicated by attaching use cases to final states. When processing flows from use case uc1 to use case uc2, the final state of the state machine description of uc1 is associated with uc2. This technique for sequencing is also used to forward processing to predefined state machines that describe re-occurring functions such as client-side validation of user data. In addition, sequencing use cases can be used for modeling use cases as the steps of a scenario-level use case [Coc2001].

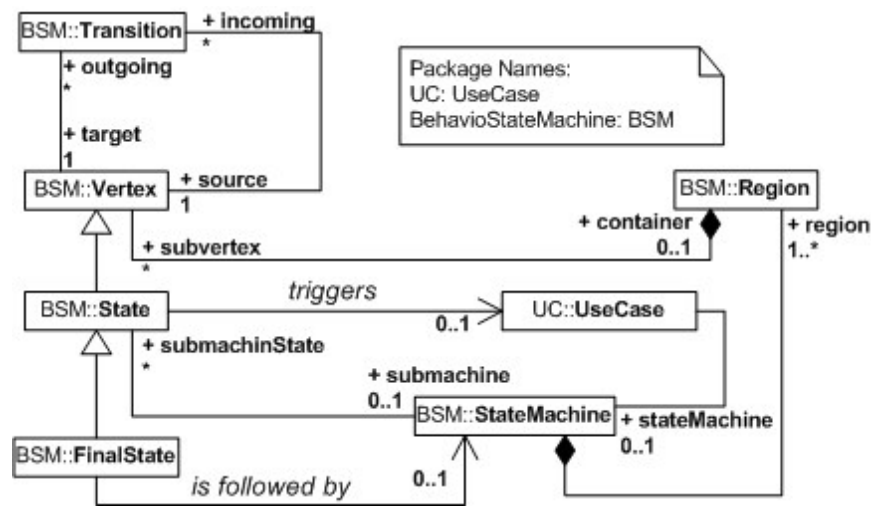


Figure 14 APSM State Machines

The APSM extends the UML specification by allowing use case associated with a state machines state. Additionally, for navigation, such as a situation where a successful completion of a task leads to another web page, it is necessary to indicate that a given use case sequentially follows another use case¹³. This sequencing of use cases is indicated by attaching use cases to final states. When processing flows from use case *uc1* to use case *uc2*, the final state of the state machine description of *uc1* is associated with *uc2*. This technique for sequencing is also used to forward processing to predefined state machines that describe re-occurring functions such as client-side validation of user data. In addition, sequencing use cases can be used for modeling use cases as the steps of a scenario-level use case [Coc2001].

¹² Note that it is a design choice whether to have a chain of use cases or a single use case with several sub-use cases. The APSM allows both without imposing one over the other.

¹³ Note that it is a design choice whether to have a chain of use cases or a single use case with several sub-use cases. The APSM allows both without imposing one over the other.

States of state machines represent the status of the application at any given time and often the status of a web application has a presentable aspect. As Figure 15 shows, states may be associated with presentations; such states are referred to as “presentation states”. A presentation - i.e. a page, form or table - is mostly composed of other UI components. Hence, a presentation is a *UI Composite*, which means it could contain other UI Components.

UI Components may be associated through a *FieldOperation*. This allows client-side actions to be defined. Examples of such actions are given in the enumeration type *ActionType*. A *UI component* is associated with a *data composite* in order to model the data support required.

Data composites are composed of data entities. A data composite can also participate in an association with another data composite, where one data composite is used as the basis for data selection from another data composite. For example, such an association would be created to model a situation where selecting a country in a component affects a list of provinces in another component. One can see from Figure 15 that APSM supports the presentation and data in one model. This is essentially modeled by an association from *UIComponent* to *DataComposite*. This association indicates that a UI component can have a supplying data composite, which is in turn, associated with data entities.

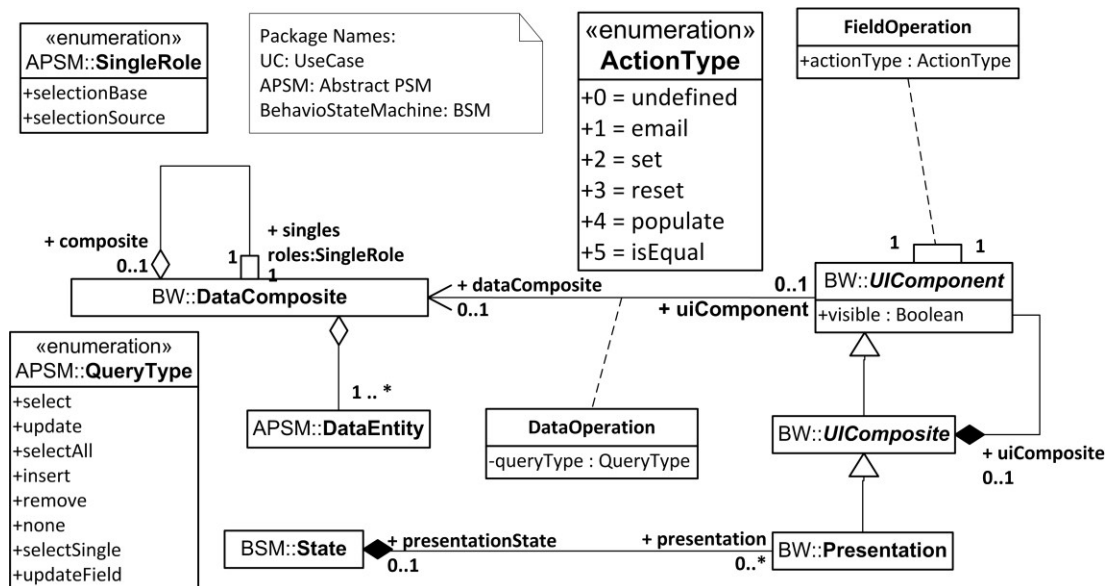


Figure 15 APSM Presentation and Data

4.4 Web Application Back-End

So far, we presented the front-end of the applications in the APSM. Figure 16 presents the part of the APSM related to events and operations handling. A controller class is used to control an application according to the behaviour defined in every use case. In addition to controller classes, one or more service classes are used to perform operations required for data access. 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 among different operation units. Controllers are a common approach to modeling the behaviour of use cases [Larman2005]. Examples of code-level controllers are Java servlets or .Net front-controllers. Similarly, services can be compared to Data Access Layer (DAL) files in .Net or Entity Beans in Java.

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. Signal events acknowledge the receipt of asynchronous messages on transitions.

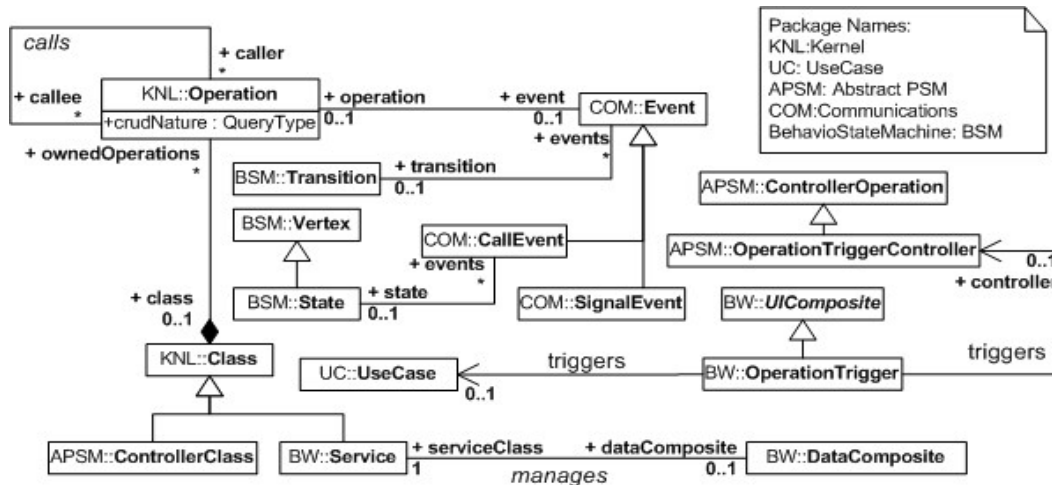


Figure 16 Events and Operations in APSM

Signal events are not always related 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 17 shows an example of such situation. In this example, the selection of a province in state *Page Shown* results in the filling of the list of towns and villages in state *Town*

List Populated. In such a case, a signal event would be attached to the outgoing transition from the state *Page Shown*.

Call Events are associated to states and transitions. They confirm an operation call and are passed to use-case controllers. A call event is required on a transition outgoing a state when an incoming transition to that state carries a signal event. This call event is used as a mechanism to call a controller operation in order to perform required tasks. The reception of the call event triggers a controller operation, which has the exact same parameters as the signal event. For server-side tasks, the controller operation refers to service operations.

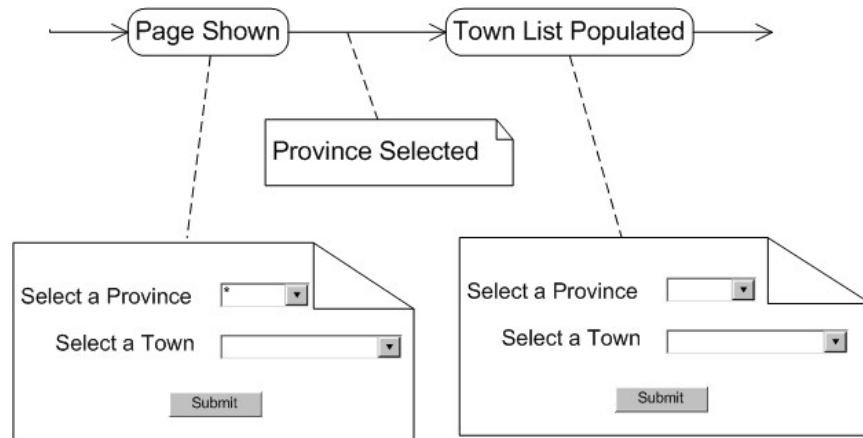


Figure 17 Signal Event created based on the selection event of a drop-down list.

A *service* is associated to a *Data Composite*. Service operations are Create-Read-Update-Delete (CRUD) operations for data composites but APSM can be extended to allow more complex CRUD operations. The extension would be in the form of a new literal added to the enumerative type *QueryType* (Figure 15). Attribute *crudNature* of the class *Operation* specifies the type of service operation based on the literals defined in *QueryType*. An association from class *Operation* to itself models the call structure from controller operations to service operations. A special *UI Composite*, *OperationTrigger*, is used to represent information submitted to the server side, for example, as part of a web form. Operation trigger may cause either an included use case or a controller operation to be triggered.

4.5 An Interactive Model

*Content-Oriented UI Elements*¹⁴, including text, image and video, may have a feedback, which could be another content-oriented UI element; that is a feedback may receive feedback as well. This is shown in Figure 18 as an association between *ContentOrientedUIElement* and *Feedback*. *Feedback* is defined as an interface to allow target platforms implementation of a specific strategy. This feature plus the fact that UI composites can recursively contain other UI composites - shown in Figure 15 - help support Web 2.0 applications. The class *ContentOrientedUIElement* is an abstract class. Figure 18 also shows the support for tagging. As shown in Figure 18, the owner of a tag and the owner of the contents are the same *User*. Every tag is associated with a tagged search that looks for all items carrying that tag.

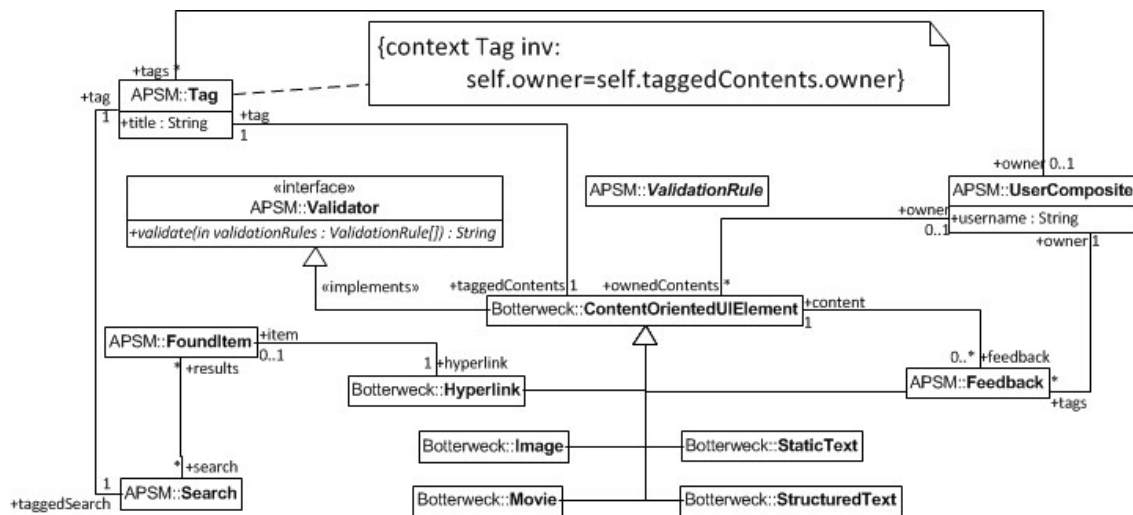


Figure 18 Making the content interactive

Figure 18 also presents a subset of APSM, which enables the validation of content-oriented UI elements. The class, *ValidationRule* is an abstract class, which might be used to validate the contents of a UI element. The validation is performed through an interface *Validator*. *Validator* has an operation, *validate* that accepts a list of validation rules as parameter. These validation rules are rules against which the *ContentOrientedUIElement* should be checked.

¹⁴ See Section 3.5.

4.6 Site Map

Figure 19 depicts a portion of the APSM corresponding to the meta-model of a site map. Site maps are useful to record different navigation paths in an application. This information could be used for analyzing the most popular paths, creating shortcuts, load balancing and error tracking.

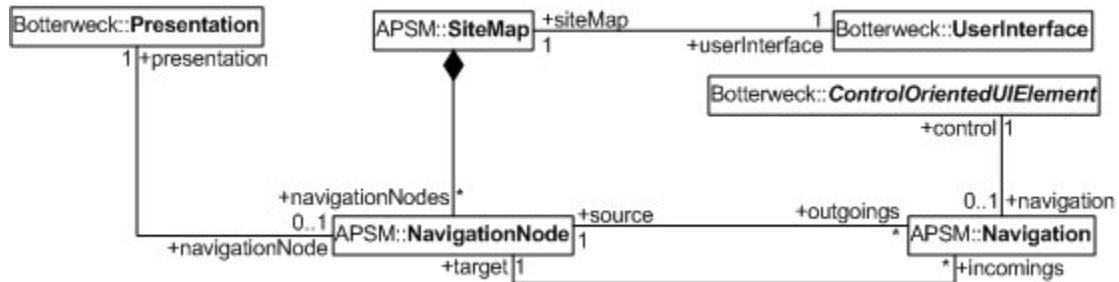


Figure 19 A Site Map Meta-Model

According to Figure 19, every user interface has a site map. Site maps are composed of navigation nodes, which are related to presentations. Navigation nodes are related through incoming and outgoing navigations. *Navigation* is associated with a *ControlOrientedUIElement*, which can be a hyperlink, operation trigger or event (See Figure 18).

4.7 Web Application Security

The APSM provides a default support to security features as presented in Figure 20. Specific modeling elements, *DataComposite*, *UIComponent*, *PageVariable* and *Service* may implement the interface *SecureItem*. The specific method to achieve security may differ from platform to platform but common approaches are as follows:

- *DataComposite* can be secured by encryption using secure keys such as session ID, username, password or IP address. It remains a design choice to secure data composites or not but since data composites are the information containers that transfer between layers there is a good chance web developers want to secure them before submission.
- *UIComponents* (and *PageVariable*) can also be secured. This might be done for terminal security as some email clients do (e.g. gmail) or to avoid accessing forged pages. If the *UIComponent* is a *UIComposite* such as a web page, an implementation option is the *https* protocol. For other *UIComponents* the same techniques as *DataComposites* and *PageVariables* may be applied.

- *Service* might need to be secured as well especially if the service is to be deployed as a remotely accessible web service. Services could be secured with respect to their access ports - using SSL for example - the encryption of their parameters and by enabling an authentication mechanism for remote invocations.

Nevertheless, the abstract model does not suggest any specific technique. The interface *SecureItem* has two operations:

- Operation *secure*, which performs the process of securing the subject of the operation. The value returned is a string, which could be the encrypted data, serialized information or the connection string to the secured port depending on the platform.
- Operation *unsecure* may be seen as the “undo” of *secure*. The return type is an *Object* in a general sense. The implementation would use an object conversion based on the type of secured subject.

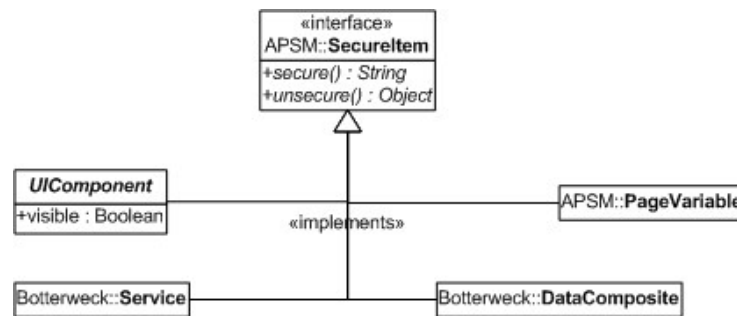


Figure 20 Securing Important Items at an Abstract Level

4.8 A Sample Authentication Model

A set of authentication use cases, classes and behaviours is provided as an example instance of the abstract model. These will be mapped to specific technologies at the target. Figure 21 depicts the supported use case diagram. Three actors are identified:

- *User* is not registered and may access the use case *Register* for registration and the use case *Activate Account* for activating his registration. Most of the existing web-based systems support a two-phase registration process, in which the Users fill out a registration form first but need to subsequently confirm the registration through email, phone or an online bank account.

- *Member* is a registered *User*. Once *Users* successfully complete the use case *Activate Account* they become *Members*. A member may access use cases *Login*, *Logout* and *View Homepage*. The use case *Activate Account* includes the use case *Login*; this means the main scenario of the use case *Activate Account* includes the scenario of the use case *Login* as well. The use case *Login* may be extended by the use case *Lock Account* after a certain number of unsuccessful login attempts. The use case *Login* also invokes the use case *View Homepage*, which involves the steps required to browse the *Member*'s personalized page.
- *Admin* is a system-side user with access to key use cases such as *Activate Account*, *Lock Account* and *Unlock Account* when manual intervention is required.

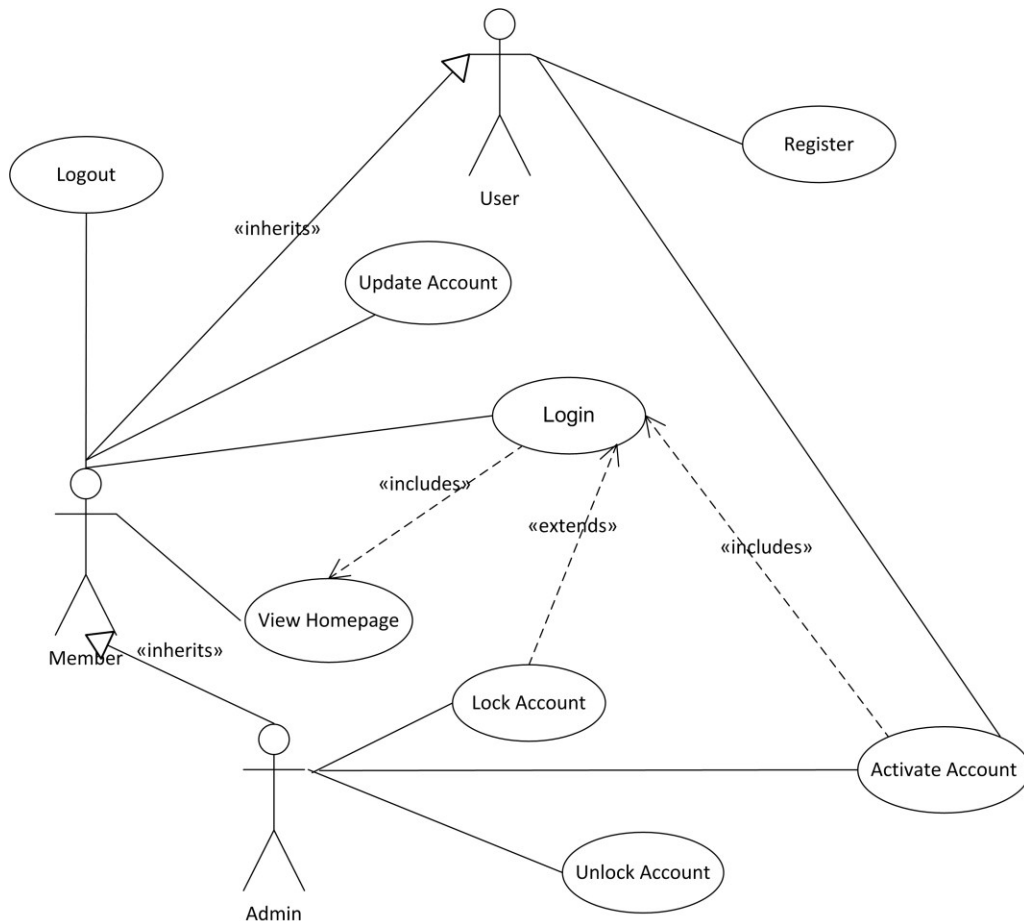


Figure 21 The Default Authentication Use Cases of the APSM

Figure 22 shows instances of the APSM classes that support Figure 21, by having every use case associated with a controller. Controllers and Services are operation-only classes. Data composites and data entities are attribute-only classes. The following dependencies exist in a typical instance of the APSM such as the one in Figure 22:

- Controller-to-Service, when a controller operation calls one or more service operations.
- Service-to-DataComposite, when a service operation has parameter(s) of type one or more data composites.
- DataComposite-to-DataEntity from every data composite to its affiliated data entity. This dependency means that methods will be required for converting data to and from the data entity.

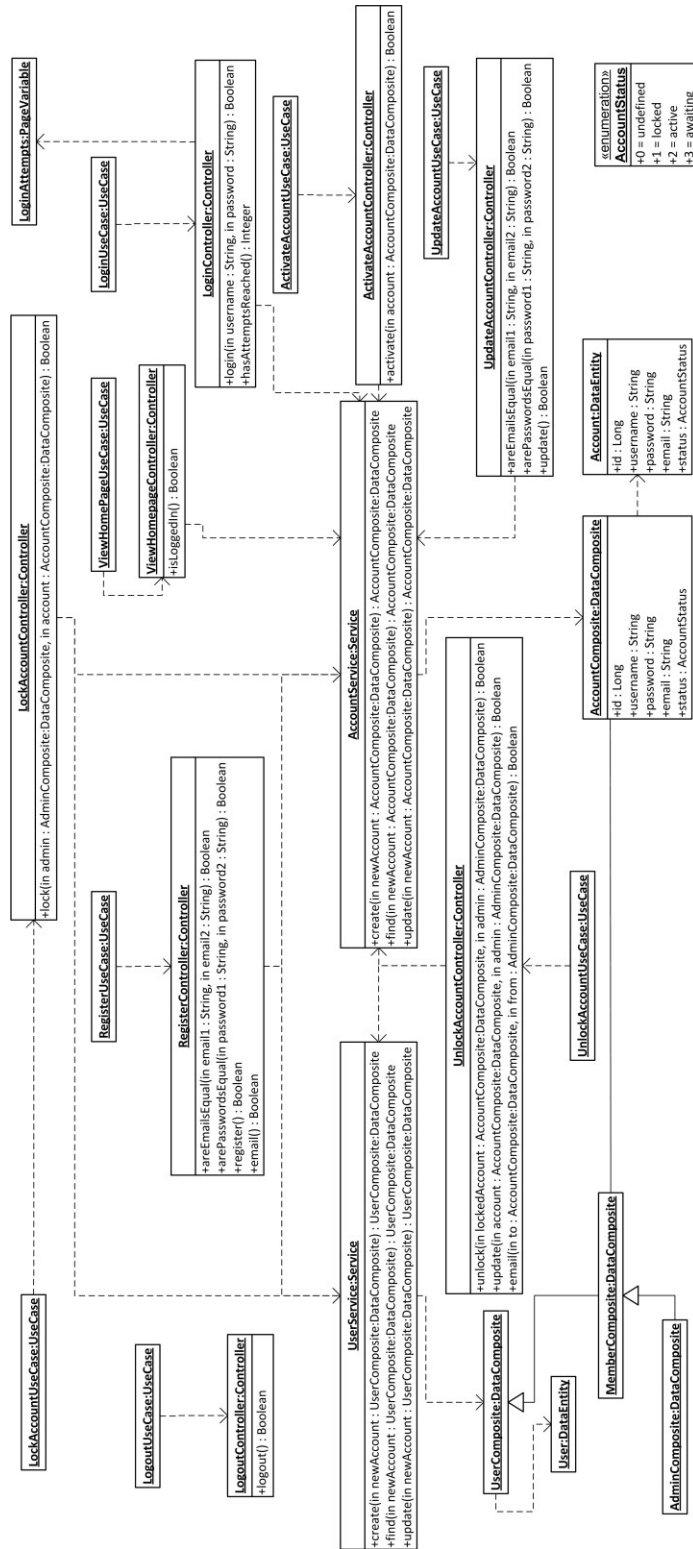


Figure 22 An Authentication Model based on the APSM

4.9 A Data-Driven Example

The APSM is particularly good for modeling data-centric applications. Figure 23 shows a model for a shopping cart application. The *ShoppingCartController* handles the operation trigger *Make Payment* submitted from a web page. The *makePaymentOperation* performs several tasks among which are an update of the inventory and a record of payment. This is modelled by having *makePaymentOperation* associated to two operations from *ProductService* and *PaymentService* as callee. These operations accomplish the required CRUD operations.

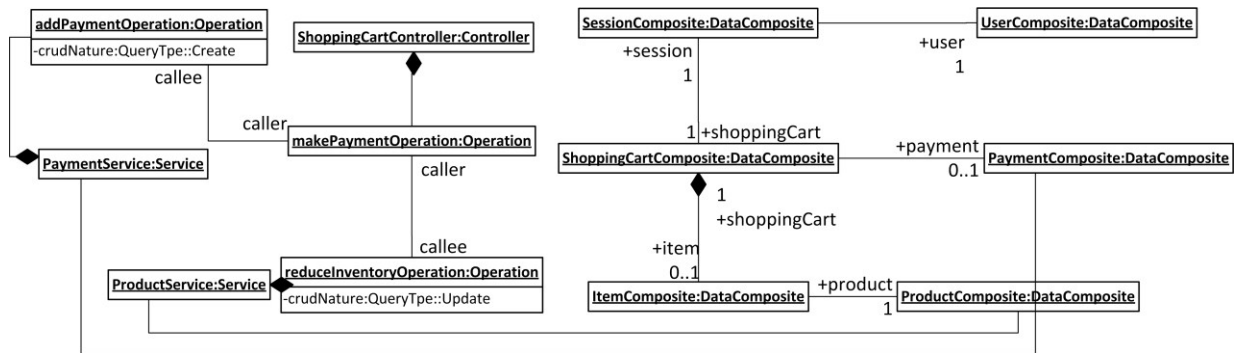


Figure 23 A Shopping Cart Example: Make Payment Operation

4.10A Multi-Layered Model

The APSM is based on the MVC architectural pattern, by the separation of controllers, presentation units and data models. The responsibility of controlling the behaviour is assigned to controllers. Presentation units can only affect controllers by raising events while controllers can affect both the data and the presentations by either directly calling operations or invoking other services. In addition, the APSM follows the controller design pattern by assigning one controller to each use case. All operations perceived by analyzing use cases are assigned to this controller. The controller, afterwards, forwards the flow to either a service operation to perform data-related operations or to another controller operation to complete a client-side task.

The APSM does not limit the number of layers. For example, the presentation layer could be divided into 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. As an example, consider an online tutorial that concerns an application, where a section of the tutorial is created for each of the application features, and a

link is provided from each section in the tutorial to its corresponding feature in the application. Thus, every feature may be invoked from either the application place or within the tutorial. A tutorial, in this example, could be seen as a scenario-level use case, as defined by Cockburn [Coc2001]. Figure 24 shows how the APSM can model this example as two interacting layers.

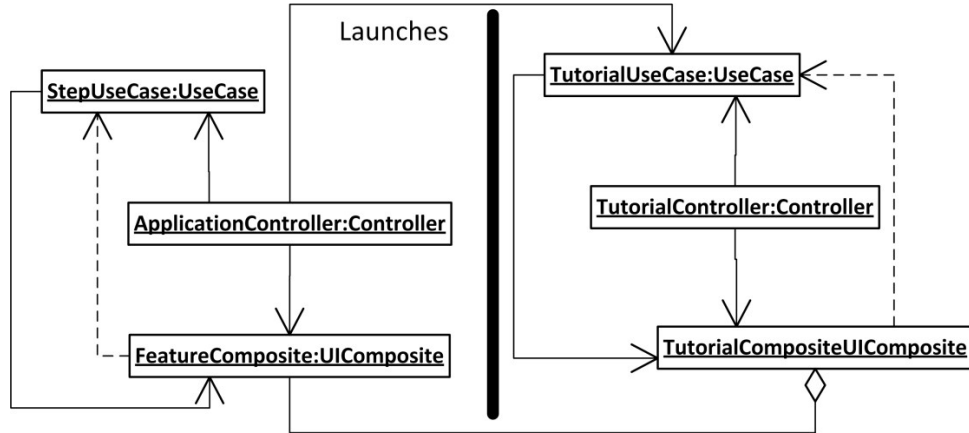


Figure 24 Using the APSM for Modeling an Application in Multiple Layers

4.11 Extending the Botterweck Model

We kept certain parts of the Botterweck model as such in the APSM. These are mainly the elements related to web application contents such as input and multimedia elements. Other parts required changes. For example, the connection of data and UI elements is only provided for elements such as *Select* components in Botterweck's model. From the viewpoint of this thesis, several other elements may be populated by data supplied from a data source; some examples are text descriptions of items, pictures, feedback on textual posts and radio buttons. Thus, we choose to assign data to all UI components.

Compared to the Botterweck model, the APSM facilitates the data access mechanism by associating UI components to data composites. Additionally the type of a data service is modelled as an association class attached to the association between a data service and an UI element, instead of having data service and UI element associated several times for many data service types (Figure 25). The modification results in one association between data service and UI element and new types can be added by updating an enumeration (*QueryType*) referenced by the association class (Figure 26). In the Botterweck model, only data-oriented UI composites such as *select lists* can be associated with input fields (Figure 25). In the APSM, any UI component can be associated to a data composite (Figure 26).

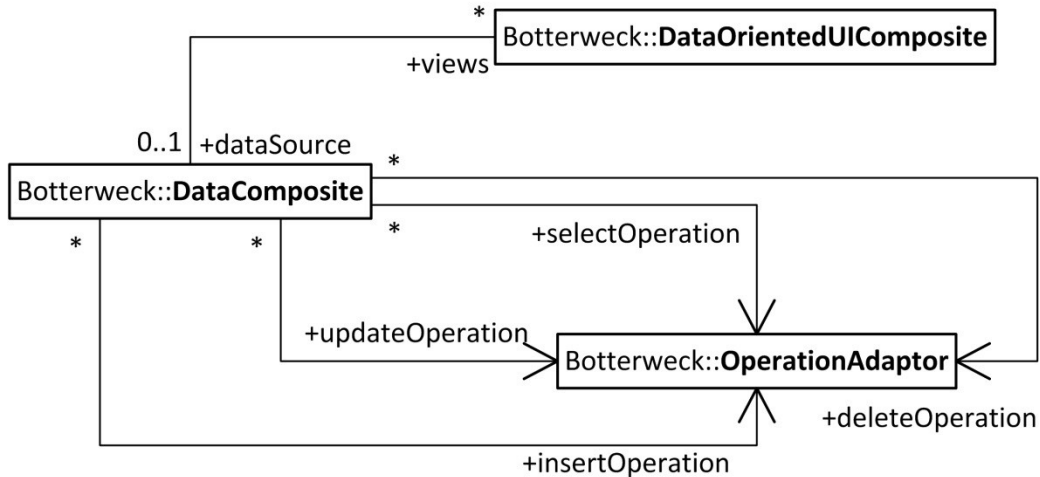


Figure 25 Association of Data Operation to UI Elements in the Botterweck Model

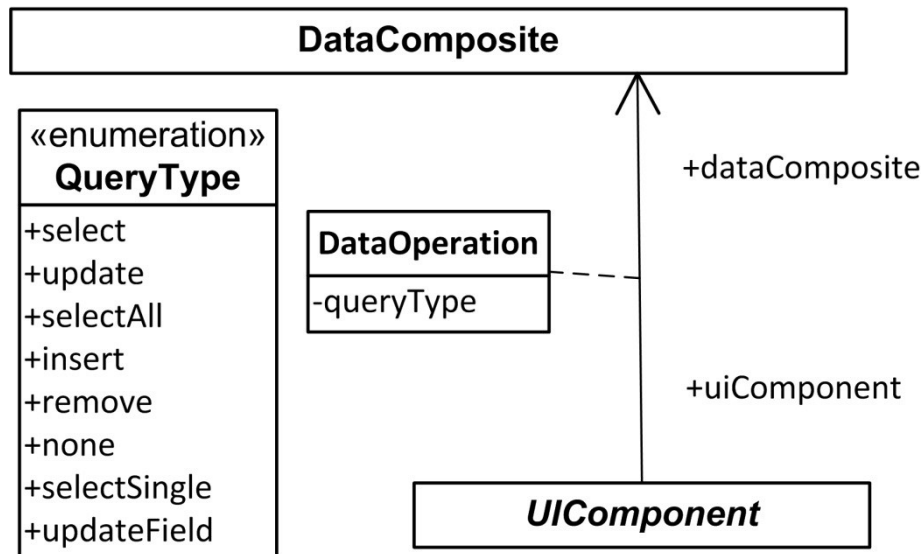


Figure 26 Association of Data Operations to UI Elements in the APSM

The possibility to associate data elements to elements such as operation triggers must be supported in order to support enhanced local storage and data orientation as required by HTML 5.0 and Web 2.0. However, the Botterweck model considers operation triggers as UI elements. In the APSM, operation triggers are inherited from UIComposite. This change allows the association of Operation Triggers like any other UI Components to data composites in the APSM. For example, a conventional web form – or an equivalent XMLHttpRequest in Ajax [Garr2007] - is submitted with the parameters – or headers - containing the value of all the UI elements enclosed within.

Another change was required to denote the order of events on a presentation unit. Presentation units can hold several kinds of events that are not necessarily operations. For example, a drop-down select UI component can cause a selection event that affects the contents of other components. In order to enable the specification of such behaviours, we have added an attribute to the association between *UIComposite* and *UIComponent* at the *UIComponent* end named *eventHolder*, which can be attributed with an *order* property. This property is only used to specify the order of such events when more than one component can cause a page to call a controller operation. Note that the association applies to all *UI Composites* and not just *Operation Triggers* to support the possibility of adding client-side events to other types of UI Composites (Figure 27).

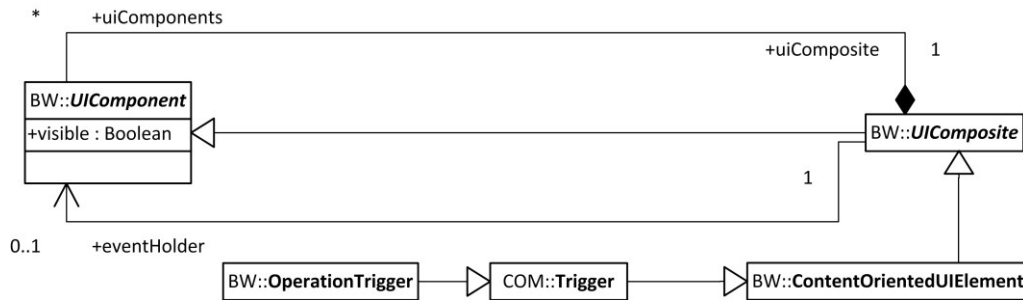


Figure 27 Operation Trigger as a UI Composite in the APSM

UML allows states to enclose other state machines. This capability could be used to fulfill a variety of objectives. A state enclosing another state machine can be interpreted as an indication of an extend/include relationship; that is the use case represented by the enclosed state machine is either an extension to the use case of the original state machine or is included within the original use case. Generally speaking, any use case attached to a state that is preceded by at least one transition carrying a *false* guard is understood as an extension. The reason is that a *false* guard denotes a branch in the regular execution of a use case. Other use cases are interpreted as inclusions. An operation trigger may also accept a reference to another use case as a denotation of an included use case.

The relationship between *State* and *StateMachine* was added to the APSM from the UML specification – as in the Botterweck model. We also added a relationship from *FinalState* to support the sequence of flows. The sequencing of state machines is indicated by the attachment of the state machine activated afterwards to the final state of its preceding use case. This is typically done for navigation reasons – such as situations where successful completion of a task

results in the navigation to another web page. Use case sequencing may also be used to specify forwarding to predefined state machines such as Login/Logout state machines that describe re-occurring functions.

We added page variables to the APSM. Page variables are session/cookie parameters or hidden form inputs typically used to carry information in a web page. They are also used as parameters for controller operations. Page variables can be used for example:

- To carry a default error message when the outgoing transition from a choice is labelled false.
- To carry the username and/or password for every transition incoming or outgoing a state, where login is required.
- To carry the number of unsuccessful login attempts.
- To hold the number of items and prices in a shopping cart.

UI components may be hidden. To this end, an attribute *visible* of type *Boolean* is added to the class *UIComponent*. In cases where a piece of information is hidden from the User but it is likely to be visible later – or vice versa - it is better to use hidden UIComponents. For example, an input field to add a new video is not by default visible on *YouTube* but once the User logs in, she can see the box and add a new video.

Certain aspects of web applications require security mechanisms to prevent insecure access to data, presentation or operations. The implementation of these mechanisms varies in different platforms. The APSM includes an interface named *SecureItem* in order to support items that may require secure access. By default the three elements, *UIComponent*, *DataComposite* and *ControllerClass* implement this interface. This may or may not lead to the usage of explicit security features in the specific platforms in accordance with the application requirements and platform-based specificities.

Other noteworthy changes include the following. An association class between a data composite and its corresponding service class was added. This association includes the type of data operation, and is required to support data storage, retrieval and processing tasks in web information systems. We also added the notion of startup use case the APSM as every application requires at least one main use case for launching. Other additions to the Botterweck

model are the notion of use case controllers for handling use case behaviour, as well as support to specific use cases such as *Login* and *Startup* and the unification of the definition of presentation states with the element *state* through an association attribute.

4.12 Summary

An abstract web model was presented in this chapter. The model is designed to answer common requirements of a web model. Hence the model supports data storage, retrieval and communication between layers. The UI is supported by an extensive collection of UI components that are capable of receiving feedback and handling logic. The flow of events is managed by UI events sent to use case controllers, which forward data-related tasks to services. While the model is used as a meta-model for defining *apsms*, a subset of the model is used for the definition of the *pims* based on a visual language; these are presented in Chapter 5 along with examples. Appendix A contains a complete description of the details of all classes in the APSM.

Chapter 5. THE PIM

A subset of the APSM is used as the PIM. The PIM is the meta-model for building *pims* used in the context of semi-automated transformations to generate *apsms*. To alleviate the process of designing *pims*, we developed a visual language that supports the elements of the PIM. This language was partly inspired by WebML [Ceri2000]. The language elements were selected based on the components found in Botterweck's model [Botter2007] with updates to conform to established features of Web 2.0 [Web2011] and HTML 5.0 [HTML2010].

5.1 A Visual Input Language

The goal of the language is to provide a more abstract and convenient way of defining models based on APSM. The elements assigned to define the visual appearance of the application are as abstract as their counterparts in the PIM to ensure the language can support all types of web applications. We have also enabled the modeling of abstract functionality and data dependencies.

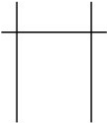
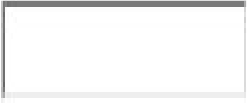
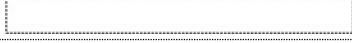
5.1.1 Components

A first group of symbols is composed of the UI components that may appear individually or in groups on presentation units such as web pages, submit forms and information tables. Table 4 lists these symbols. Elements in this table will be used within UI composites, which could be translated to different things such as page or window depending on the specific platform.

Table 4.(a) Component Symbols of a Web language

Symbol	Component
	Submit Button
	Check Box
	Date
	Link
	Password
	Plain Input

Table 4.(b) Component Symbols of a Web language

Symbol	Component
	Single Select
	Text Message
	Table Column
	Text Area
	Image
	Video
	Radio Buttons
	Time
	Voice
	Formatted Text
	Progress Bar
	Static Message
	Cart
	Multiple Select

5.1.2 Information Elements

Two symbols presented in Table 5, are used for information elements. The first symbol represents a piece of data on the page, which is to be communicated to the server as parameter of a CRUD operation. In order to guarantee the abstractness of the UI level, the language does not make distinctions between a single entity, a processed subset of entities or a combination of multiple entities. The second symbol defines a variable; that is a piece of information shared by

several units. This could be a session or page variable. For sake of abstractness, the language does not differentiate between different types of variables.

Table 5. Information Elements

Symbol	Type of the Element
	Data
	Variable

5.1.3 Logic

Logic is defined using the operator symbols in Table 6. These operators are used towards both the data and variable elements shown in Table 5.

Table 6. Logic Symbols

Symbol	Operation
+	Create Data or Set Variable
<<	Populate UI Component with all instances found within data or variable
<<?	Populate UI Component with some of the instances found within data or variable
-	Drop Data or reset Variable
?	Query Data without loading it
>>	Update data
++	Add all items or entities from a data/variable set to a composite
@	Email that sends an email to the email address provided by a variable attached to the operation.
=	Are the values of two components equal?
↓↑	Sort a table collection in given direction.

The order of operations is as follows:

1. Operations in the same use case are first ordered based on the flow that the state machine mandates.

2. Operations in the upper-level presentations have higher priorities; that is an operation defined in a presentation p has a higher priority than the one defined in a sub-presentation of p .
3. The order of operations within the same level is understood by their numbered labels. Therefore, an operation labeled 1 has a higher priority of an operation labeled 2.
4. Operations with no numeric label do not have a priority. Thus, if the presentation has more than one operation without numeric label, the order would be random.
 - a. If a presentation has only one operation, the operation does not need to be numbered.

5.1.4 UML

A subset of UML is used to depict state machines as listed in Table 7.

Table 7. UML	
Symbol	Type of the Element
	Actor
	Choice
	Final State
	Initial State
	State
	Note
	Transition

5.1.5 Semantics of a *pim*

A visual *pim* has a state machine, one or more actors and a UI prototype. A UI prototype is composed of one grouping symbol. A grouping symbol is composed of zero or more grouping

symbols as well as zero or more UI components, logics and data. A logic represented in a visual diagram can be interpreted based on the arrows that connect different UI, data and logical components. Every arrow has one or more source and target components, and involves an operator from Table 5. The source components are the input providers, while the targets of the arrows are consumers of the output. The operator expresses how data is manipulated. Table 8 provides some examples with their meaning.

Table 8.a. Examples of the Visual Language and their Semantics

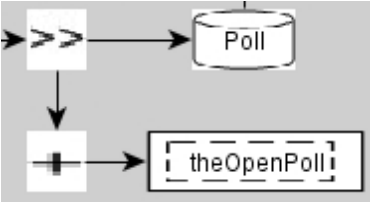
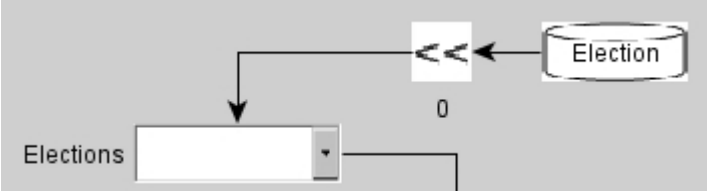
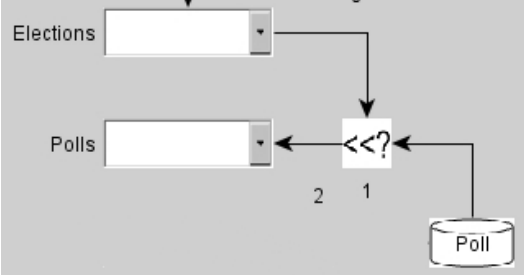
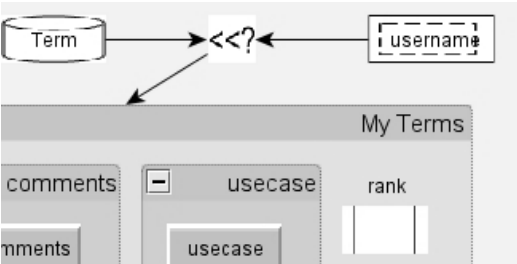
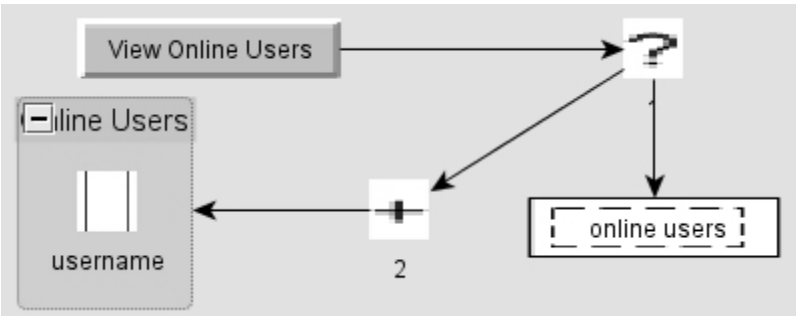
Visual <i>pim</i> Example	Semantics
	<i>Login</i> submit button provides parameters to find entries from <i>User</i> data source.
	The updated <i>poll</i> from <i>update</i> operator (>>) is used to set <i>theOpenPoll</i> using the set operator (+).
	The <i>Elections</i> select component is populated by the data coming from the <i>Election</i> data source.
	Data from <i>Poll</i> data source is used to populate the <i>Polls</i> select component. The selected entry from <i>Elections</i> is also used as another input to the <i>selectSome</i> operation (<<?).
	The data from the <i>Term</i> data source is used to populate the <i>MyTerms</i> table. The contents of the <i>username</i> page variable is used as another input – that is to filter the data for the <i>selectSome</i> operator.

Table 8.b. Examples of the Visual Language and their Semantics

Visual <i>pim</i> Example	Semantics
	The contents of the form containing View Online Users submit button is used to find (?) data from the online users page variable. Found data is used to set the Online Users table (+).

5.2 PIM as the Meta-Model of the Visual Language

Figure 28 shows the subset of the APSM used to specify the visual input language. This model constitutes the meta-model of the *pims*. The elements of the PIM include what the end User expects to see and communicate with, i.e. user interface and data requirements. State machines and their related classes are included in the *pims* because they describe the User experience in terms of steps that a User takes to accomplish a use case. UI components are also included as they are the main elements that a User communicates with. We also include Data composites because they represent contents provided to Users. Finally, client-side operation types (*ActionType*) as well as CRUD operations (*QueryType*) are. The semantics of classes and their associations is the same as in the APSM.

Figure 28 uses abbreviated package names as follows:

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

Table 9 lists the mapping between elements from the PIM and the visual symbols.

Table 9.a.The mapping between the PIM and its visual language

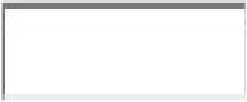
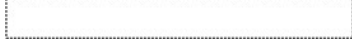
Graphical Symbol	PIM Element
	DateField
	Hyperlink
	PasswordField
	InputField
	SelectMany
	SelectOne
	StaticText
	DataColumnView
	InputArea
	Image
	Video
	SelectOne
	TimeField
	Voice
	StructuredText
	PageVariable

Table 9.b.The mapping between the PIM and its visual language

Graphical Symbol	PIM Element
	DataComposite
+	insert/set
<<	selectAll
<<?	selectSingle
-	remove/reset
?	select
>>	update
++	populate
@	email
=	equal
	sort
	Actor
	Choice
	Final State
	Initial State
	State
	Note

5.3 Examples

Examples in this section are taken from two case studies: Dilmaj [Dilmaj2010] and the Account Management System (AMS) [AMS2010]. The case studies are elaborated in Appendix B and Appendix C respectively. The first example is chosen from Dilmaj that is a Web 2.0 application, while the second example is selected from the AMS.

5.3.1 Example 1: Adding Comments to a Content-Oriented UI Element

Figure 29 shows the input model of the use case *Add Comment* defined using the language of Section 4.3. Figure 30 shows an object model instantiated from APSM and corresponding to the input model in Figure 29. The model involves three of the application use cases: *Add Comment*, *View Comments* and *View Dilmaj*. The use case, *View Comments* is attached to the state *Add Comment* to realize the inclusion relation from the use case, *Add Comment* to the use case, *View Comments*. The use case, *View Dilmaj* sequentially follows use case *View Comments* as expressed by the attachment of use case *View Dilmaj* to the final state of the use case, *Add Comment* state model. Note that there is a concrete link between every pair of name-dependent objects. This addresses issue 3 in Section 3.1 about name-based dependencies. The object model also shows that the input from *Add Term*, the table *Terms Table* and the button *Close* are all considered independent parts of the presentation to ensure compatibility with Web 2.0. Transitions and actors are not included for brevity.

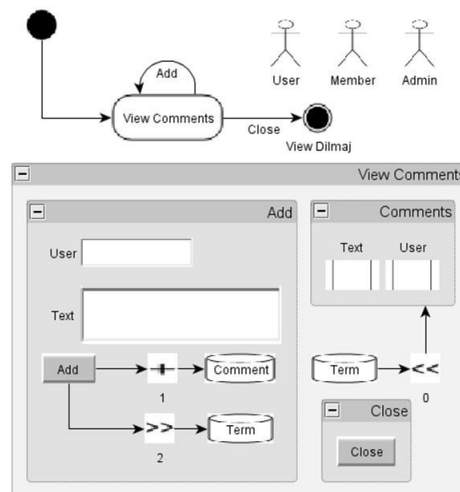


Figure 29 The pim of the Use Case, Add Comment

Figure 31 shows an object model containing the modelling elements that handle the behaviour of the application. *AddCommentController* is dependent on both *TermService* and *CommentService*. It calls four operations;

- *addComment* to create a new comment and to attach it to the selected term
- *updateTerm* to update the corresponding term accordingly
- *close* to hide the popped up comments window, and

- *populateCommentsTable* to load existing comments.

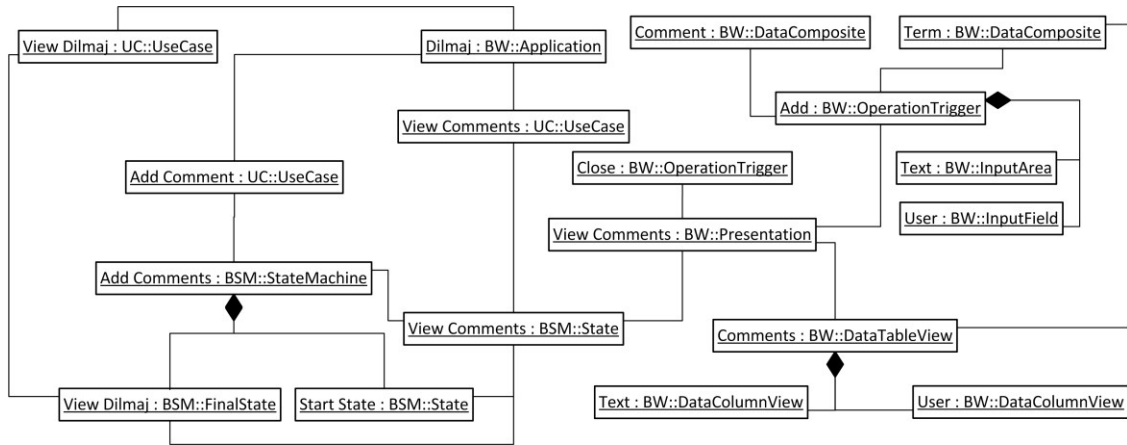


Figure 30 The Object Model of the Use Case, Add Comment

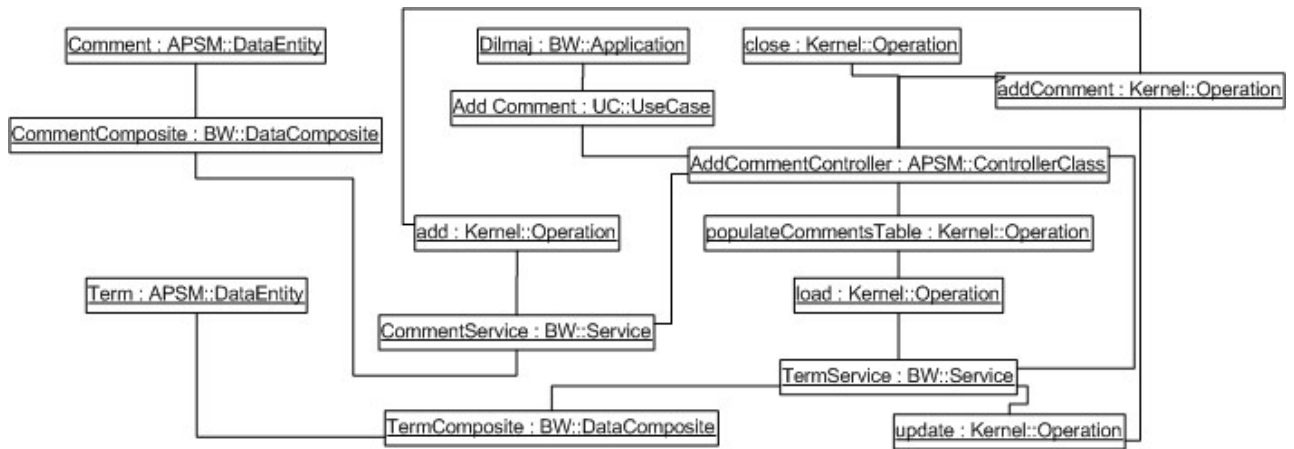


Figure 31 The Object Model of Behavioural Features of Figure 29

5.3.2 Example 2: Sequencing State Machines

In this example, the focus is on the support APSM provides to the relationships between two use cases using a session variable. The first use case, *View Bills* specifies browsing through the list of previous bills while the second one, *View Bill* shows a selected bill. Figure 32 shows the UI model of use case *View Bills* while Figure 33 shows the UI model of *View Bill*. Figure 34 and Figure 35 show the *apsm* automatically generated for these two use cases. The page variable *selectedBill* plays a key role in the model. It is set at the last step of the state machine *View Bills*, and used as shown in Figure 35, in the first step of the state machine corresponding to *View Bill*

to populate the bill display panel. The gap between the models in Figure 32 and Figure 33 and the ones in Figure 34 and Figure 35 is bridged by the transformations described in Chapter 6.

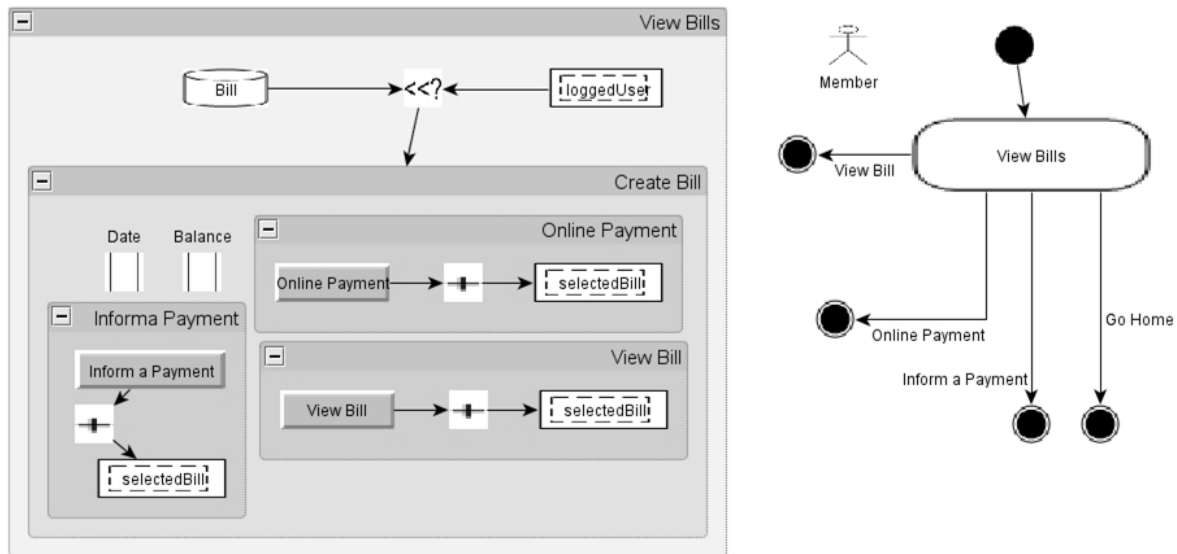


Figure 32 The *pim* of the Use Case, *View Bills* of the AMS

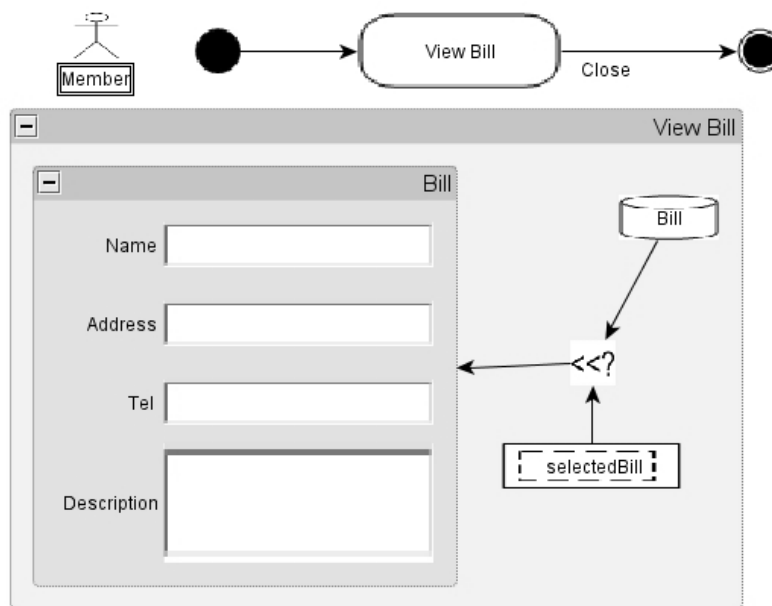


Figure 33 The *pim* of the Use Case, *View Bill* of the AMS

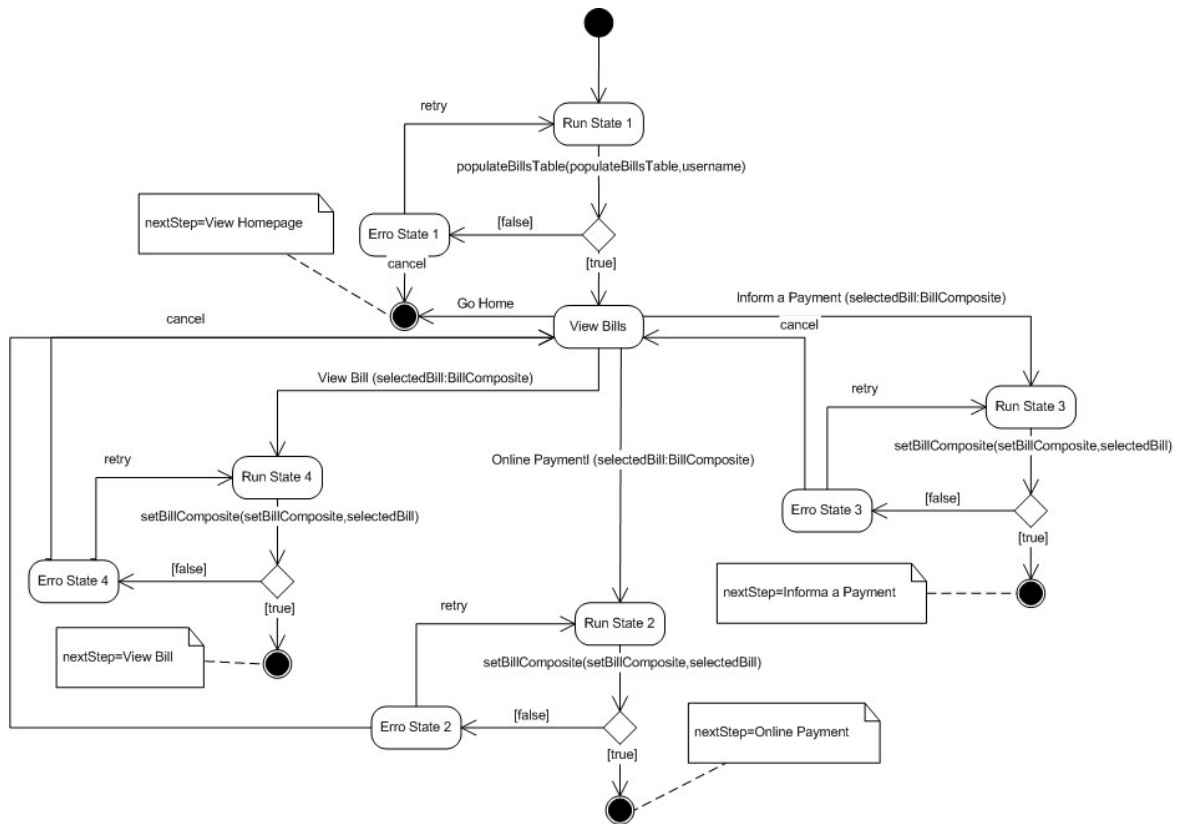


Figure 34 The *apsm* of the Use Case, *View Bills* of the AMS

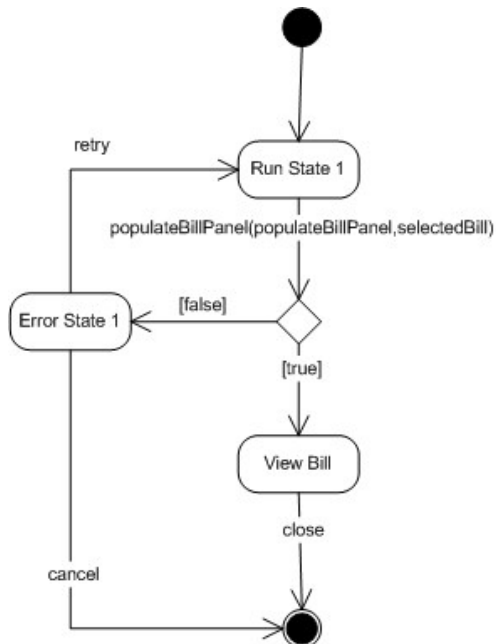


Figure 35 The *apsm* of the Use Case, *View Bill* of the AMS

5.4 Summary

In this chapter, we presented a visual language for defining *pims*. Such *pims* are to be used as source of automated transformations that lead to complete *apsms*. The PIM is a subset of the APSM, in which the features related to server-side tasks, events and controlling states of the applications are omitted. The components of the PIM were selected based on the existing models and technologies.

Chapter 6. **MAPPING THE PIM TO THE APSM**

In this chapter, we present transformations for mapping the PIM to APSM. These transformations map an input model such as the ones in Figure 29, Figure 32 and Figure 33 to an abstract web model such as the ones in Figure 30, Figure 31, Figure 34 and Figure 35. Since an objective of model-driven development is to automate parts of the software engineering process, it is important to show that it is possible to derive the full specification of an application from a minimum set of descriptions, i.e. a *pim* defined using the visual language.

6.1 PIM-to-APSM Mapping Rules

In our proposed approach, the PIM and the APSM are both defined based on the same meta-model. The PIM is defined as a subset of the APSM shown in Figure 28. The PIM-to-APSM mapping discussed in this chapter takes a *pim* and adds the required details to build an abstract web application according to the APSM. This mapping is an example of refining mappings defined in Section 2.6.3. The PIM-to-APSM mapping ensures that the elements of the APSM used to define the usability and data requirements (the PIM subset), are properly integrated with the ones specifying the behaviour and data access mechanisms.

Figure 36 presents a summary of the information used from the PIM as well as that created at the APSM level. In this figure, circular items are the inputs that are to be mapped to rectangular ones, i.e. outputs. Solid arrows are direct mapping rules whereas dashed ones indicate target elements that are indirectly inferred. Direct mappings are one level mapping rules that create an element with the same attribute/value pairs as the one found in the source. Indirect ones however, need more levels of analysis and require more complicated rules. Here is a summary of the mappings found in Figure 36:

- The data associations from the UI model are used to infer data objects and navigation paths through data objects.
- The contents and structure of web pages are determined by the contents of the presentation states and transition flows.

- Transitions and states from the input model are also used to create events, operations and controllers.
- The generated events, operations and controllers are used in turn to build the data access services in combination with data associations from presentation states. They are also used to map parameters to attributes within the data model.

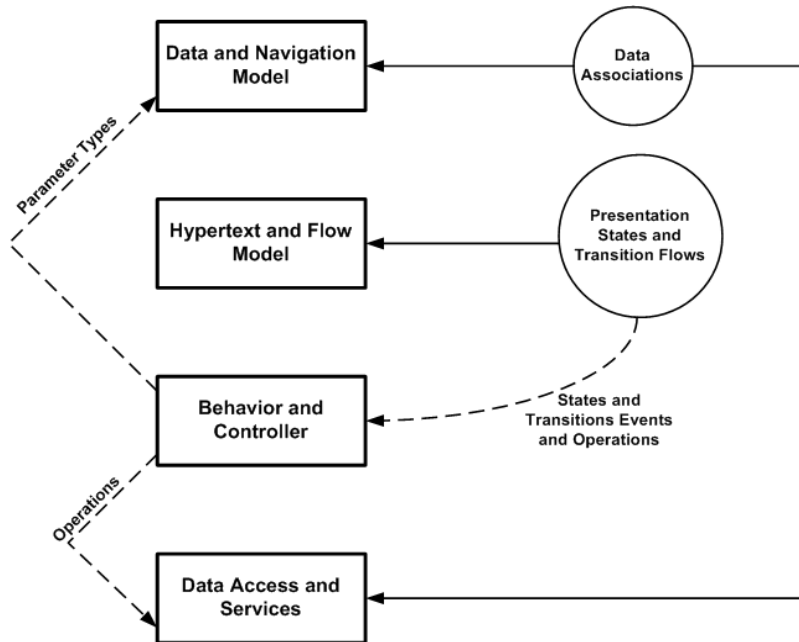


Figure 36 Summary of PIM-to-APSM mappings

As a running example, consider the use case *Open Poll* from the case study *Election Management System (EMS)*. Figure 37 shows the *pim* of the *Open Poll* use case and Figure 38 shows the corresponding object model. Figure 39 shows the *apsm* state machine of the *Open Poll* use case and Figure 40 contains the classes involved in the *apsm* of the *Open Poll* use case.

Figure 37 shows that *Open Poll* use case has an actor *Admin*. This use case is composed of one presentation state *Open Poll*. Transitions outgoing this state end at two final states each attached to other state machines, *voting* and *welcome*. The figure also denotes that *Open Poll* presentation is contained in another presentation named *homepage* that is *Open Poll* is a unit on the web page *homepage*. *Open Poll* contains two main triggers, *open* and *logout*. *Open* contains most of the input elements including the *Elections* and *Polls* select. The *Elections* select is the target of an operation that loads all elections, and the source of another operation that selects some of the polls. This means the selected election will affect the selection of polls and also that

the definition of poll and election are associated. Triggering *open* updates the loaded poll and also sets the page variable *theOpenPoll* with the updated poll.

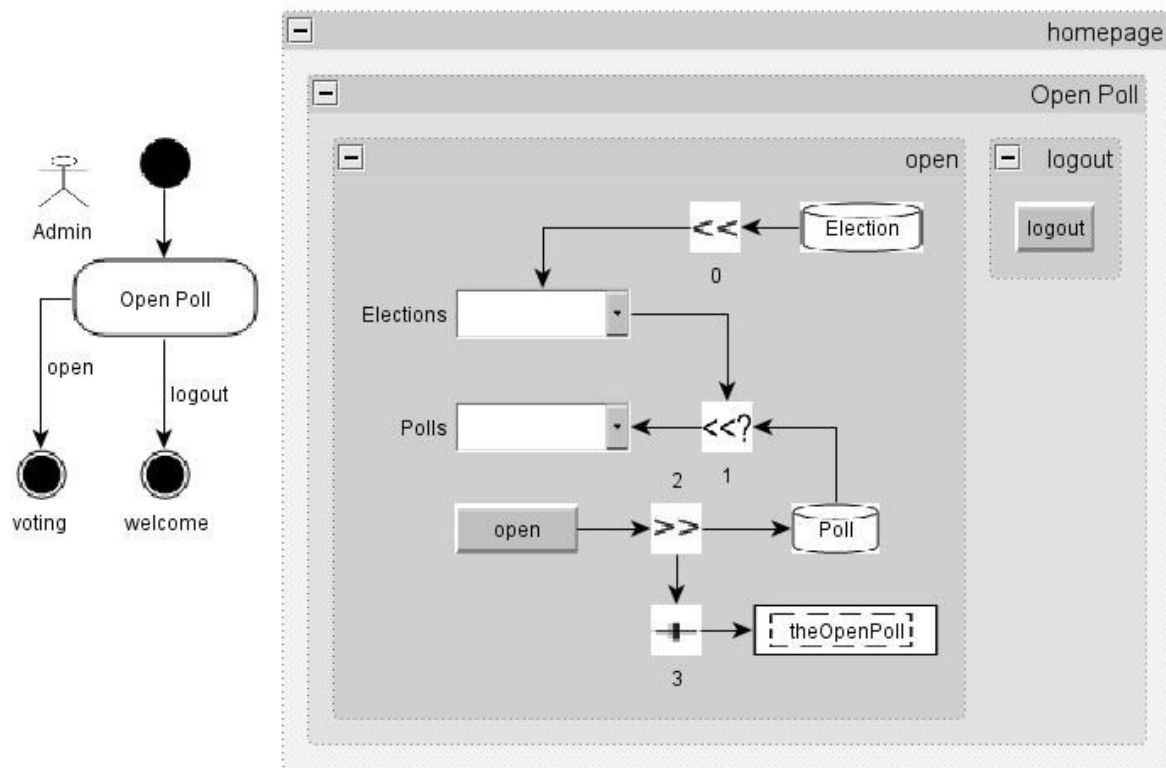


Figure 37 The *pim* of the *Open Poll* use case

Figure 38 presents the object model of the *Open Poll* use case. Note that data operations are presented as instances of the class *Logic*. There are four instances of this class labelled as 1, 2, 3 and 4 using UML notes. The attribute *order* on these instances specifies the order of running the logic starting from 0. Instance 1 denotes an operation of type *selectAll* from the source *Election* to the target *Elections*, which is the *SelectOne* component in Figure 37. Instance 2 has two sources: one is the data composite *Poll* and the other is the *Elections* select component, which will be used as the key to select some of the polls. Instance 2 populates the *Polls* select component. Instance 3 has one source only, which is the operation trigger meaning that the whole data found within trigger will be used as the source. One target is the data composite *Poll*, which will be updated. Another target is the instance 4 of the *Logic*. Instance 4 uses the provided input to set its own target, the page variable *theOpenPoll*.

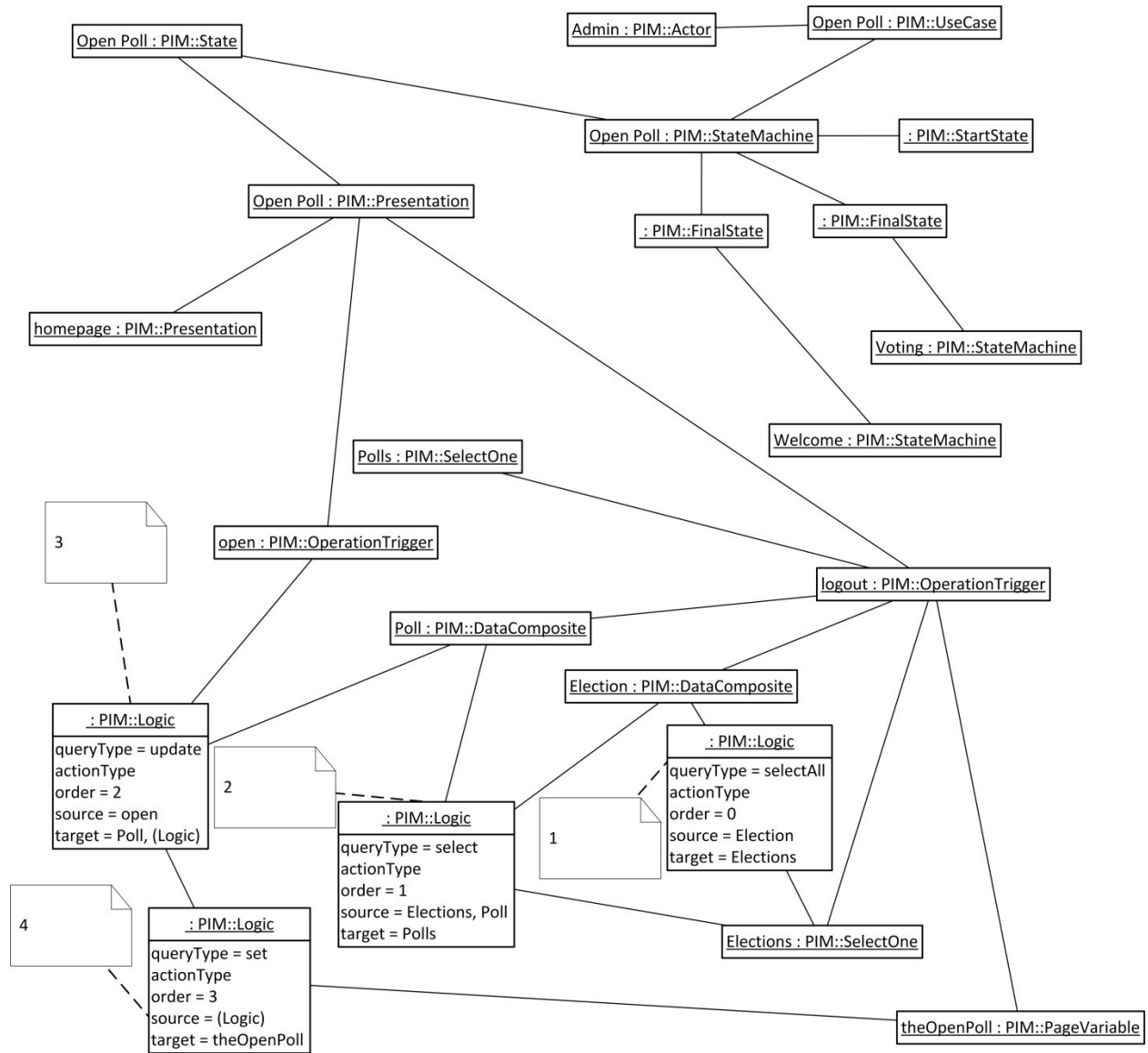


Figure 38 The Object Model of the *pim* of the *Open Poll* use case

Figure 39 shows the *apsm* version of the state machine in Figure 37. According to Figure 38 four *Run States* are added, which invoke the required events by calling operations. Run states are typically followed by a *choice* that verifies the returning value from calling the operation. If the operation returns *false*, the path is directed to an *Error State*. Error states are presentation states that provide *retry* and *cancel* options. The *retry* transitions return to the previous *Run State*. The *Cancel* transitions return to the previous presentation state if any or to a final state that has a *nextStep* other than the one associated with the regular path of the state machine – that is the path going through all *true* transitions.

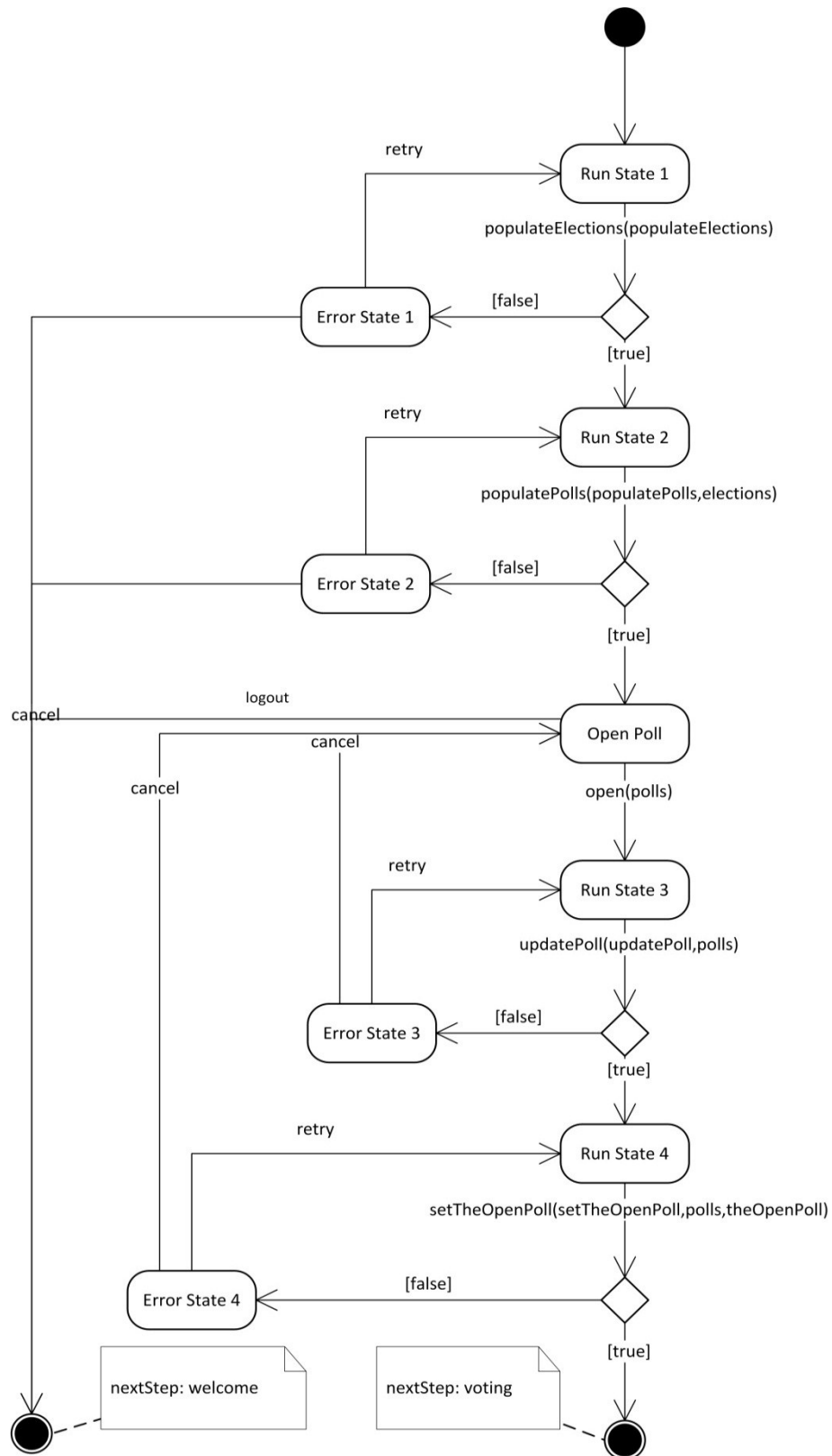


Figure 39 The *apsm* State Machine of the *Open Poll* use case. First parameter of every event is the operation to call.

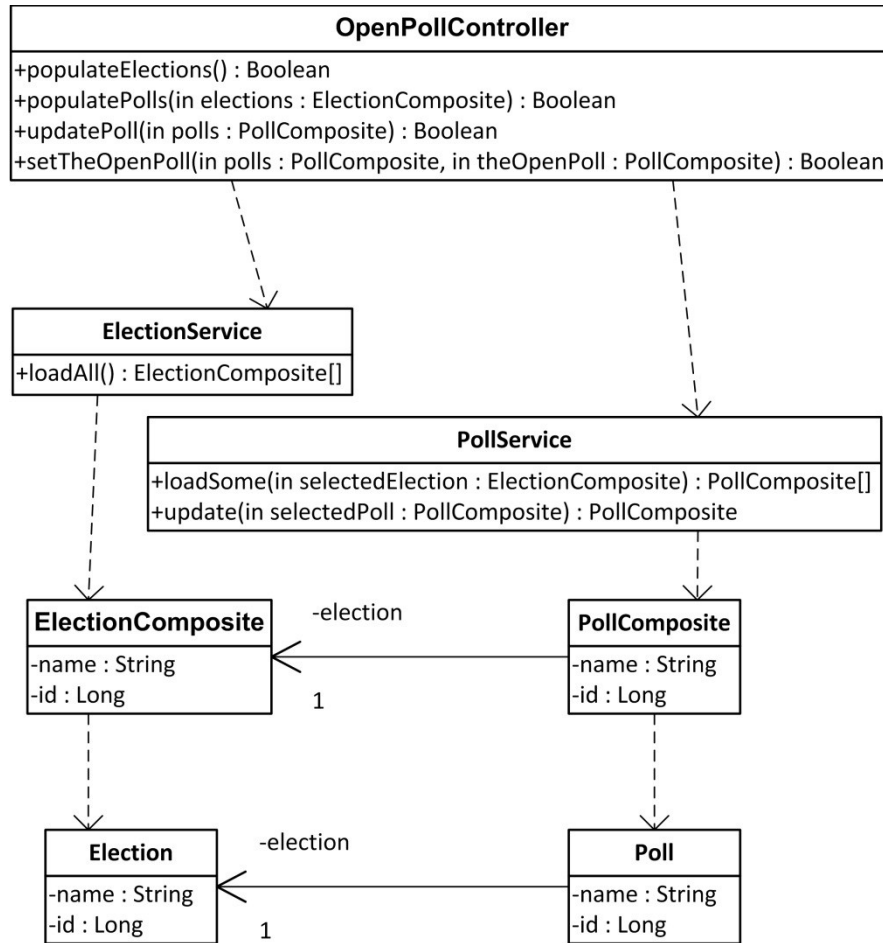


Figure 40 The *apsm* Classes of the *Open Poll* use case

Figure 40 shows the *apsm* classes related to the *Open Poll* use case. Sections 6.1.1 to 6.1.6 elaborate the mapping rules that take models like the one of Figure 36 and transform it to the ones of Figure 39 and Figure 39.

6.1.1 Mapping Data Entities

An entity is generated for each data composite associated with an operation trigger in a presentation. UI components contained in the presentation are mapped to entity attributes of the created entity class. Text inputs become *String* attributes. Selectable components such as lists and drop-downs are mapped to *Long* attributes representing the ID of selected elements. Date and time components are mapped to *Date* attributes. And linked contents such as hyperlinks, and multimedia are mapped to *String* attributes that will hold the source of the media.

In classes *Poll* and *Election* from our running example, attributes *PollID* and *electionID* are added (Figure 41) because of a drop-down input with the same name as the entities on the

page. The association is created because the *Election* drop-down provides the input used to populate the *Polls* drop-down (Figure 36).

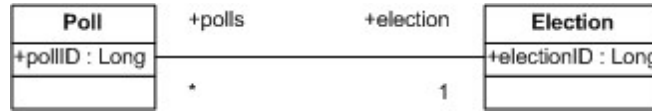


Figure 41 Entity classes generated for the state machine of Figure 7.

6.1.2 Mapping Signal Events

Given an outgoing transition triggered by a *submit* button, a signal event is created with the same name as the submit button. 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 42 shows an example of such situation.

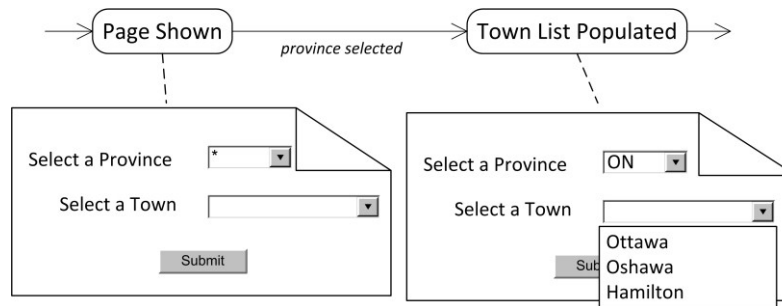


Figure 42 Signal Event created based on the selection event of a drop-down list.

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 order number of the logic operation to decide which event occurs first. For example, in Figure 37, the order of the *selectAll* operation applied to the *Elections* select is 0 suggesting this would be the first operation to take place.

6.1.3 Mapping Call Events

Call Events are added to transitions to call a controller operation. Each operation found in a presentation state is mapped to a call event. For example, four call events in Figure 37 are created to handle four operations found in Figure 39 (*populateElections*, *populatePolls*, *updatePoll*, *setTheOpenPoll*). The core of this mapping consists of a state named *Run State* followed by an order number (See Figure 43). This state is followed by a choice to check

whether an error has happened during the operation call. The choice has two outgoing transitions labelled *true* or *false*. The *false* labelled transition ends at a presentation state named *Error State* followed by the same order number as the *Run State*. *Error States* have two outgoing transitions, one carrying a *retry* event and another carrying *cancel*. *Retry* returns to the *Run State*. *Cancel* returns to the previous presentation state if any. But if there is no presentation state preceding the *Run State*, the *Cancel* transition leads to a final state attached to the previous state machine. For example, in Figure 38 *Cancel* outgoings of *Error States 1* and *2* end at the final state connecting to the *Welcome* state machine, which shows the first page of the application; and *Cancel* outgoings of *Error States 3* and *4* return to the *Open Poll* state. The *true* and *false* on outgoing transitions of choices are guards. The *Cancel* and *Retry* on outgoing transitions of *Error States* are signal events. Note that all call events have a first parameter. This parameter is the controller operation to be called. All parameters except for the operation name are mapped to the controller operation.

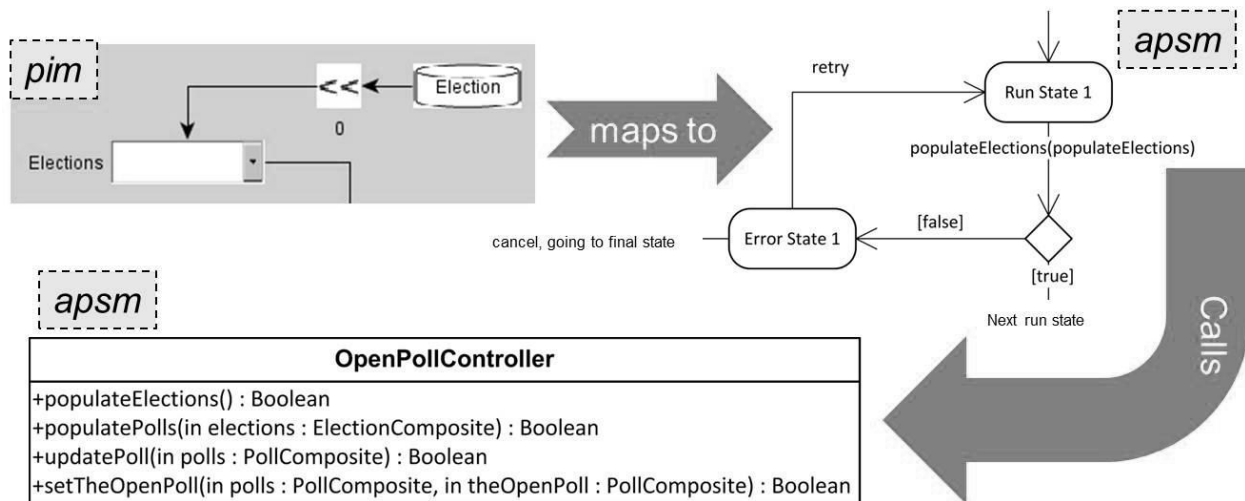


Figure 43 *selectAll* Operation of Figure 37 mapped to a Call Event in Figure 39

6.1.4 Mapping Controller Operations

A controller class is created for each use case. The responsibilities of this class include grasping front-end operations, performing the required logic treatments and issuing data access calls. The most common form of controller operation is the one created for handling a signal event. These controller operations are triggered by call events and use values obtained from the mapping of UI components as parameters. Each controller operation returns a *Boolean* value.

Controller operations are named according to the following conventions. An operation that aims to control 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 the first word.

As an example, to support the operations found in the *pim* of Figure 37, the following operations are added to the controller as shown in Figure 40:

- *populateElections():boolean* – This operation loads all elections from the data store in the list box.
- *populatePolls(ElectionComposite elections):boolean* – This operation uploads from the data store all Polls that are dependent on the provided election parameter.
- *openPoll(PollComposite polls):boolean* – This operation changes the status of a selected poll to open.
- *setTheOpenPoll(PollComposite polls, PollComposite theOpenPoll):boolean* – This operation sets the page variable *theOpenPoll* to the selected poll from the *polls* select component.

6.1.5 Mapping Services

A service is associated with a data composite. Service operations are automatically generated for every CRUD operation required by a data composite. This is shown in Figure 44. According to Figure 43, a service class *PollService* is created based on the data composite *Poll* found in the *pim* and a service class *ElectionService* is created based on the data composite *Election*. Figure 43 also shows that for every CRUD operation found in a *pim*, a service operation is added to the service class corresponding to the data composite. 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 (See Figure 39).

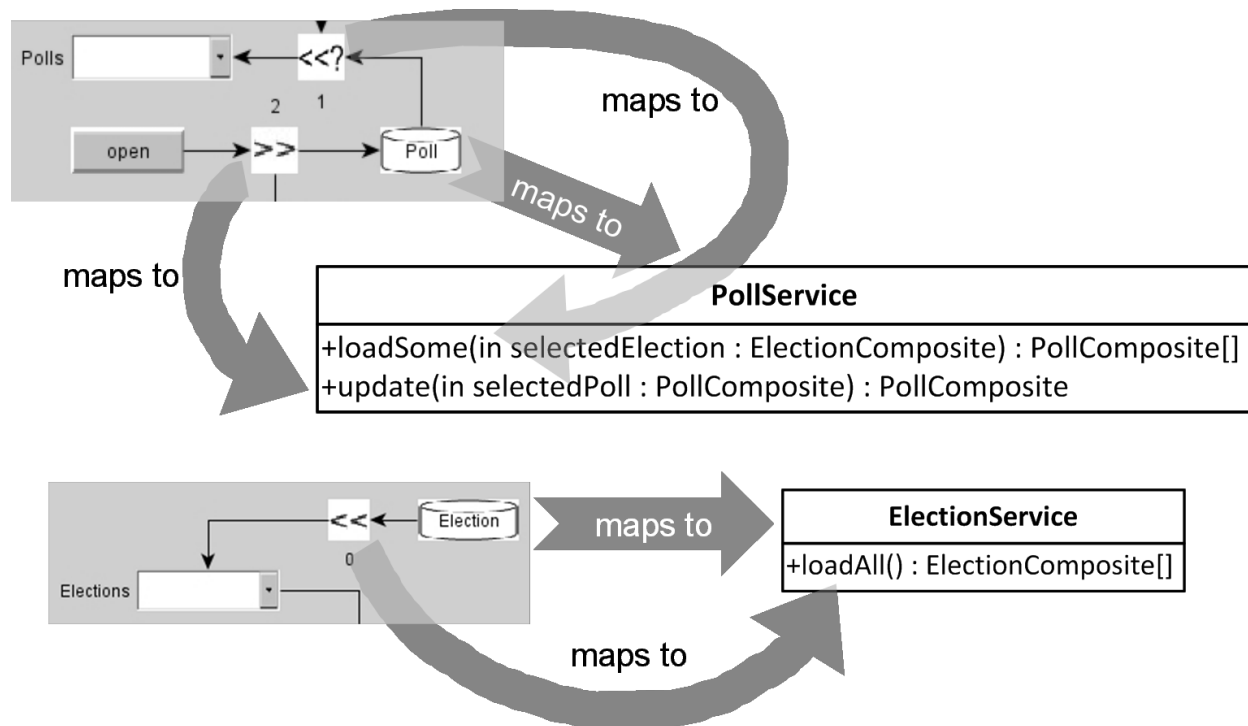


Figure 44 CRUD Operations of Figure 37 mapped to Services in Figure 40

6.1.6 Mapping Domain Objects

Domain objects are inferred based on data element associations in UI models. The data composites found on the *pim* are used to indicate the name of the domain objects. The attributes of domain objects are also mapped from the UI components found within the same model. UI components become *String* attributes. A *select* component results in an association to the source object from which the select component is populated. For instance, in the *Open Poll* example, the domain object *Poll* is associated with the domain object *Election* because a selected election is used as source for populating the list of polls (see Figure 45). A default *String name* and a *Long id* attribute are added to all objects.

As shown in Figure 44, for every domain object, a Data Composite and a Data Entity are created. The former represents the object used to carry information as parameters while the latter is only used to perform database/file level CRUD operations. Data composites are dependent to data entities because they need methods for converting them to and from data entities. Also, service classes are dependent on data composites because they use them as the types of parameters.

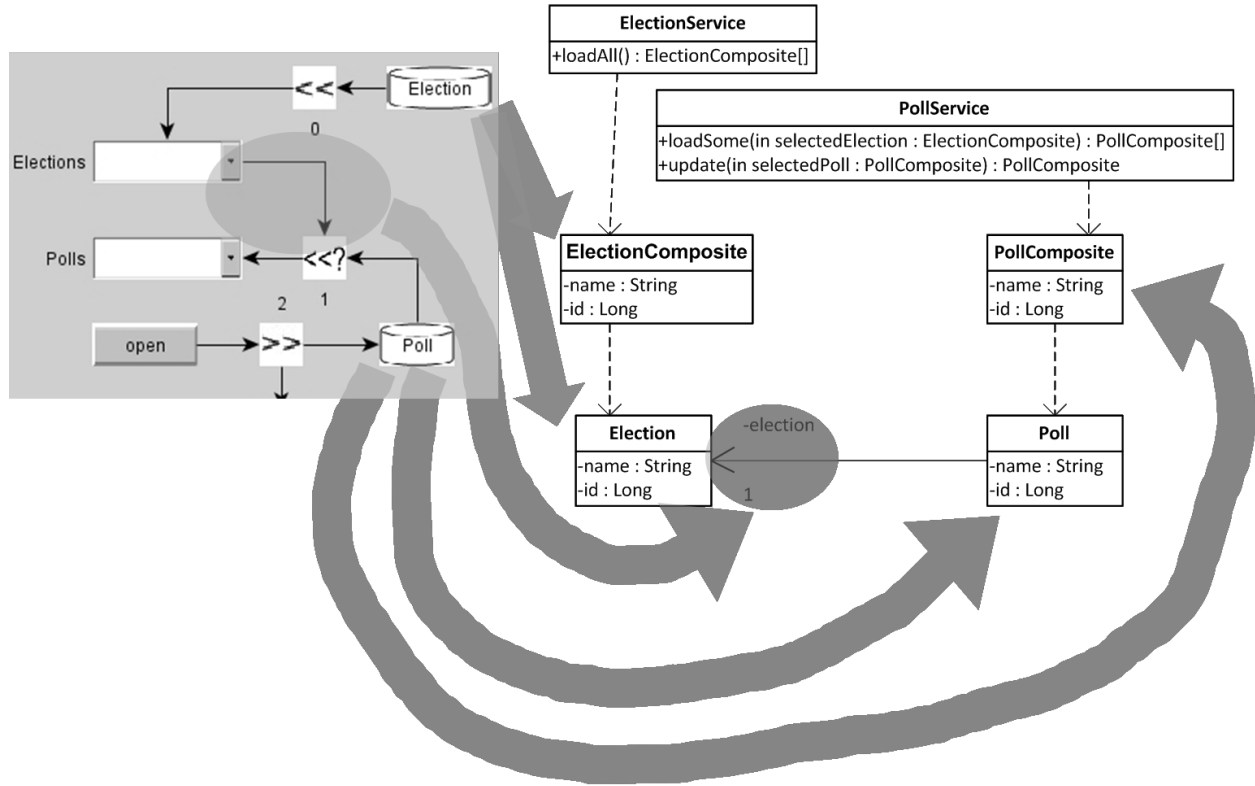


Figure 45 The *pim* of Figure 37 mapped to Data Composites/Entities in Figure 40

6.2 PIM-to-APSM QVT Relations

The PIM-to-APSM transformation is implemented as a set of QVT relations, key definitions and functions. This transformation is available in [MDWSQVT2011]. The transformation accepts two models as input:

1. *pim* is a model defined based on a meta-model, which is in fact the subset of the APSM used to define the PIM as shown in Figure 28. In general, elements from *pim* are checkonly in this transformation, which means no change is made to the input.
2. *apsm* is a model defined based on a meta-model, which is in fact the APSM as specified in Chapter 4. APSM elements are generally *enforced* in this transformation, which means, whenever applicable, changes are made to an *apsm* as the output of the transformation.

The transformation assumes that an application is uniquely identified by its name. It is also assumed that State Machines, Use Cases, Controllers, Services, Data Composites, Data Entities and Actors are uniquely identified by names within an application. Hence, we use the

names of these elements as keys. Other elements are specified uniquely with a combination of their name and the element(s) containing or limiting them as described in Table 10.

Table 10. APSM Elements Unique Keys

Element	Key Combination
State	Name, State Machine
Operation	Name, Class
Transition	State Machine, Source, Target
Signal Event	Name, Transition
Page Variable	Name, Transition
Parameter	Name, Behaviour (i.e. Operation or Event)
Operation Trigger	Name, Presentation
Presentation	Name, (presentation) State

There are nine top relations that are entry points of the validation process (see Section 2.6) in the PIM-to-APSM transformation. For every instance of the PIM, all or some of these relations may be true and hence become a separate validation track as follows:

1. *application_application* maps a *pim* application to an *apsm* one. This chain of relations is followed by the relations *usecase_usecase* and *stateMachine_stateMachine* that create the relevant use cases and state machines at the APSM level. The states of state machines are also mapped from *pim* to *apsm*.
2. *presentationState_presentationState* is a top relation that does not need any other relation as its post-condition. This relation maps presentation states to the *apsm*.
3. *hyperLink_hyperLink* is another single top relation, which maps *pim* hyperlinks to *apsm* transitions. Consider a hyperlink contained in a *pim* presentation state *s1* that leads to the state machine *sm1*. Also, consider the APSM state *s2* as the mapping of *s1* and *sm2* as the APSM mapping of *sm1*. Hence, the *apsm* transition – mapped from the *pim* hyperlink - originates from *s2* and ends at a final state, which has *sm2* attached to it.
4. *finalState_finalState* is a top relation that ensures the integrity of the mapped final states.

5. *first_presentationAssociations_signalEvents* is the top relation that maps an operation contained in a *pim* presentation. The relation creates a signal event on the transition outgoing from the presentation state; this transition ends at a new state. This new state has an outgoing transition with a call event that calls a controller operation. This controller operation, if necessary, includes a call to data service as well. Relation *first_presentationAssociations_signalEvents* is one of the most complex relations of the PIM-to-APSM transformation. It necessitates the validation of several other relations as its post-condition:
 - a. A relation to map the details of the operation trigger (such as the UI components associated with it) to the equivalent *APSM* signal event parameters.
 - b. A set of relations to map the details of the operation association to the *APSM* call event. There are different versions of such relations to map a call event based on the type of operation as well as their source and targets. For example, mapping a *create* event against a data composite requires different operations compared to the operation for adding a row to a table.
 - c. If the operation association is the last one being processed, the relation *first_presentationAssociations_signalEvents* would require ending the transition at either a final state or the next presentation state. Otherwise, the relation *next_presentationAssociations_signalEvents* (with the same structure as *first_presentationAssociations_signalEvents*) would be repeated for the next operation association.
6. *emptyOperationTrigger_signalEvent* is validated only if an operation trigger does not contain any UI components and operation associations.
7. *dataComposite_dataEntity* maps data composites to their corresponding data entities.
8. *application_siteMap* creates a site map of the *apsm* based on the navigation path specified as *pim* transitions, triggers and links.

9. *operationTrigger_operationTrigger* maps the UI components of operation triggers to the *apsm* triggers.

Beside relations, other important elements of a transformation are functions. Functions are used mainly to make decisions for cases, where both the input and output are selected from a single domain, e.g. *apsm*. These functions are:

1. *getAttributeName* that receives the name of a component and transforms it to be used as the name of an attribute by trimming all the spaces and making the first letter of all words capital except for the first word. For example, the name of an attribute obtained based on an input field titled *Admin Last Name* would be *adminLastName*
2. *addTransition* is used to add a transition to the collection of incoming or outgoing transitions of a state.
3. *getParameterType* used to decide the type of a parameter resulting from the mapping of a UI component. This happens in situations such as when the contents of a UI component are passed as parameters to a signal event.
4. *addUseCase* adds a use case to the collection of an actor's use cases.
5. *addOperation* adds an operation to a class.

6.3 A Sample Relation

The relation *first_presentationAssociations_signalEvents* is listed in Listing 4 to Listing 12. This relation creates an *APSM* signal event, corresponding parameters and required transitions for the first logic state found within a *PIM* presentation.

Listing 4 Checking the PIM

```
1:top relation
2:first_presentationAssociations_signalEvents {
3:stateName:String;
4:machineName:String;
5:orderNumber:String;

6:checkonly domain wpim uiAssociation1:wpim:UIComponentAssociation {
7:    isFirst=true,
8:    owner=
9:        operationTrigger1:wpim::OperationTrigger {
```

```

10:         presentation=presentation1:wpim::Presentation {
11:             presentationState=state1:wpim::State {
12:                 name=stateName,
13:                 outgoing=outgoing1:wpim::Transition {
14:                     target=target1:wpim::State {}
15:                 },
16:                 stateMachine=stateMachine1:wpim::StateMachine {
17:                     name=machineName
18:                 }
19:             }
20:         },
21:     },
22:     order=orderNumber
23:};

```

- Lines 1-2 declare the relation. Being a top relation means this is one of the entry points at which the validation process begins.
- Lines 3-5 declare the variables required to map the names of states and state machines as well as data associations. The variable *orderNumber* is a special variable that is used to indicate the running order of operation associations. This number is used to label the states that are automatically added at the APSM level such as the state *runState1* in Figure 39.
- Line 6 checks the existence of the association with the UI.
- Line 7 ensures that the association is the first in the list.
- Lines 8-11 span the ownership hierarchy to reach the owning state.
- Line 12 sets the variable *stateName*.
- Lines 13-15 obtain the outgoing transitions and their targets.
- Lines 16-18 reach the owning state machine and set the variable *machineName*.
- Line 22 sets the variable *orderNumber*.

Listing 5 Creating the APSM Main State

```

24:enforce domain aPSM state2:apsm::State {
25:    stateMachine=stateMachine2:apsm::StateMachine {
26:        name=machineName
27:    },
28:    name=stateName,
    ...

```

- Lines 24-28 declare the target state that matches the original presentation state in the source. Then the name and the *stateMachine* property of this state are set.

Listing 6 Transition Outgoing the Main State and an Extra Logic State added here

```

29:outgoings=outgoing2:apsm::Transition {
30:stateMachine=stateMachine2,
31:source=state2,
32:target=state2_1:apsm::State {
33:    incomings=addTransition(state2_1.incomings,outgoing2),
34:        stateMachine=stateMachine2:apsm::StateMachine {
35:            name=machineName,
36:            states=stateMachine2.states->append(state2_1)
37:        },
38:        name='runState1',
    ...

```

- Lines 29-38 handle the creation of a new state in the target without an equivalent in the source.
- Line 29 creates the transition outgoing from *state2* that was created in line 24 (Listing 5).
- Lines 30-31 set the *stateMachine* and the source properties of the transition.
- Line 32 declares the new state, *target2_1* as the target of the transition.
- Lines 34-35 assign the *stateMachine* attribute of the new state.
- Line 36 adds this state to the state machine.
- Line 38 sets the name of the state, which matches a state with the same name, *runState1* in Figure 39.

Listing 7 Transition Outgoing the Extra State and the Target Choice are added here

```

39:    outgoings=outgoing2_1:apsm::Transition {
40:        stateMachine=stateMachine2,
41:        source=state2_1,
42:        callEvents=callEvent2:apsm::CallEvent {
43:            owner=outgoing2_1
44:        },
45:        target=choice1:apsm::Choice {
46:            statMachine=stateMachine2:apsm::StateMachine {
47:                name=machineName,
48:                states=stateMachine2.states->append(choice1)
49:            },

```

```

50:     incomings=addTransition(choice1.incomings,outgoing2_1),
51:     name='choice'+orderNumber,

```

- Lines 39-51 create the transition outgoing from the state, runState1, the call event to handle the operation and the choice that evaluates the return value of the operation:
- Lines 39-41 create the outgoing transition and define its state machine and source values.
- Lines 42-44 define the call event.
- Line 45 defines a choice as the target of the transition.
- Lines 46-49 assign the state machine value of the choice and then append the choice to the states collection of the state machine.
- Line 50 adds the transition from line 39 to the incoming transitions of the choice.
- Line 51 sets the name of the choice based on the value of the variable orderNumber.

Listing 8 Transition Outgoing the Choice and the ‘True’ Outgoing

```

52: outgoing2_1_1:apsm::Transition {
53:     stateMachine=stateMachine2,
54:     source=choice1,
55:     guard=gurad21:apsm::Guard {
56:         name='true'
57:     }
58: },

```

- Lines 52-58 define a transition outgoing from the choice that carries the guard *true*. That is, this transition would be taken when the return value from the operation called by the event of lines 42-44 (Listing 7) returns a *true* value.

Listing 9 The ‘False’ Guard and the Target Error State

```

59: outgoing2_1_2:apsm::Transition {
60:     stateMachine=stateMachine2,
61:     source=choice1,
62:     guard=gurad22:apsm::Guard {
63:         name='false'
64:     },
65:     target=state2_1_2:apsm::State {
66:         stateMachine=stateMachine2:apsm::StateMachine {
67:             name=machineName,
68:             states=stateMachine2.states->append(state2_1_2)
69:         },

```

```

70:     incomings=addTransition(state2_1_2.incomings,outgoing2_1_2),
71:     name='Error State',
72:     presentation=presentation2:apsm::Presentation {
73:         name='Error Page',
74:         uiComponents=errorMessage2:apsm::StaticText {
75:             name='Error Performing Operation'
76:         }
77:     },

```

- Lines 59-77 create another transition outgoing from the choice that is guarded by the value *false*. This transition leads to another state in line 65, which shows an error message, this error message is created in lines 74-76. This state has two outgoing transitions.

Listing 10 The Retry Links from the Error State

```

78:     outgoing=outgoings2_1_2_1:apsm::Transition {
79:         stateMachine=stateMachine2,
80:         source=state2_1_2,
81:         signalEvent=signalEvent21:apsm::SignalEvent {
82:             transition=outgoings2_1_2_1,
83:             name='retry'
84:         },
85:         target=state2
86:     },

```

- The first transition is created in lines 78-86, where a transition with a signal event, *retry* is added. This transition leads to the main presentation state *state2* this is equivalent to the state, *Open Poll* in Figure 39.

Listing 11 The Cancel Link from the Error State ending at the Final State

```

87:     outgoing=outgoings2_1_2_2:apsm::Transition {
88:         stateMachine=stateMachine2,
89:         source=state2_1_2,
90:         signalEvent=signalEvent22:apsm::SignalEvent {
91:             transition=outgoings2_1_2_2,
92:             name='cancel'
93:         },
94:         target=finalState2:apsm::FinalState {
95:             name='finalState',
96:             stateMachine=stateMachine2:apsm::StateMachine {
97:                 name=machineName,
98:                 states=stateMachine2.states->append(finalState2)
99:             },
100:             incomings=addTransition(finalState2.incomings,outgoings2_1_2_2)

```

- Lines 87-100 define the second transition outgoing from the error state. This transition carries another signal event *cancel* and leads to a final state.

Listing 12 The Post-Conditions and Aftermath Relations

```

101:      where {
102:          dataAssociation_callEvent(uiAssociation1,callEvent2) or
103:          fieldAssociation_callEvent(uiAssociation1,callEvent2);
104:      operationTrigger_signalEvent(operationTrigger1,outgoing2);
105:      if (not uiAssociation1.next->isEmpty()) then
106:          next_presentationAssociations_signalEvents(uiAssociation1.next,outgoing2_1_1)
107:      else if (not target1->oclIsTypeOf(wpim::FinalState)) then
108:          trueChoice_nextState(target1,stateMachine2, outgoing2_1_1)
109:      else presentationAssociations_finalState(stateMachine1,stateMachine2,', outgoing2_1_1)
110:      endif
111:    endif;
112:  }

```

- Finally, lines 101-110 check the post-conditions of the relation:
- Line 102 checks the creation of the call event through the relation *dataAssociation_callEvent*.
- Line 103 runs the same validation but this time for an association between data and a UI field rather than operation trigger.
- Line 104 runs a relation to map the operation trigger to a signal event. This is necessary because the current relation does not create signal events at the target.
- Line 105 checks if there is more than one item associated with the operation trigger. If yes, the *next_presentationAssociations_signalEvents* relation is called to repeat the same mappings for the next association.
- Lines 106-107 map the target state of the true transition outgoing from the choice to either a final state if the target of the main presentation state is a final state in the source model; or a regular state otherwise.

6.4 Summary

In this chapter, we presented the mappings required to build an *apsm* from a *pim*. The resultant *apsm* is still independent of specific platforms but is enriched with events, services, domain objects, new states and operations required to accomplish the tasks defined in the *pim*. We

presented the mapping rules using a running example. We also provided a sample QVT relation part of the implemented transformation. The full set of relations as well as key definitions and functions can be found in [MDWSQVT2011].

Chapter 7. **TRANSFORMING TO SPECIFIC PLATFORMS**

In this chapter, four different platforms are defined based on different technologies and the transformations to those platforms are formalized. Transformations are performed to map the APSM to platforms defined based on J2EE (AndroMDA), WebRatio, Google Web Toolkit, and .Net. The definition of these platforms, mapping the APSM to them and the extensive sets of QVT relations written to support those transformations are distinctive aspects of our work. Following is a brief list of the mappings:

1. APSM-to-AndroMDA: This mapping creates a fully detailed AndroMDA model, which may be transformed to code using AndroMDA code generator.
2. APSM-to-WebML: The target in this mapping is a specific configuration of the WebML-based tool, WebRatio.
3. APSM-to-GWT: maps the APSM to a platform, based on Google Web Toolkit. This allows one to evaluate the Web 2.0 modeling capabilities of the APSM.
4. APSM-to-Net: maps the APSM to a platform based on Microsoft .Net.

These mappings are expressed using QVT relations [QVT2008]. In order to formalize the relations, four platform-specific models have been designed to specify the target platforms. These models are only developed in order to implement the transformation process. They may not be complete or exact in regard to the official documentation of all or parts of the technologies used for each of the specific platforms. Full implementation of the mappings in this chapter can be found in [MDWSQVT2011].

7.1 APSM-to-AndroMDA

AndroMDA [Andro2007] 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. A state machine state represents presentation or behavioural features of a use case. Every use case is controlled by a controller, and states and transitions events are deferred to that controller's operations. 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. The following are some definitions regarding this platform¹⁵:

- A deferrable event is an event that invokes a controller operation. Deferrable events are used to assign call events from states or transitions to operations.
- A value object is an object that carries information between domain objects and the presentation or data access layer.
- *FrontEndView* is a state stereotype used for states that represent web pages.
- *Manageable* is a tag added to an entity class meaning that entity is subject to either of *Create*, *Delete*, or *Update* operations.
- A signal event is an event that is 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 its corresponding outgoing transition.
 - b. A deferrable event is created on the target state of its corresponding outgoing transition. If the next vertex is a choice then the deferrable event is created on the transition ending to that choice.
 - c. A controller operation is generated to be called by the generated deferrable event.

¹⁵ These are not official AndroMDA definitions. They are based on our experience with it.

- 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 the 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 its corresponding 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.

7.1.1 APSM-to-AndroMDA QVT Relations

The transformation accepts two models as input:

1. *apsm* is a model instance of the APSM as specified in Chapter 4. The *apsm* elements are generally *checked only* in this transformation.
2. *andromda* is a model instance of the meta-model shown in Figure 46.

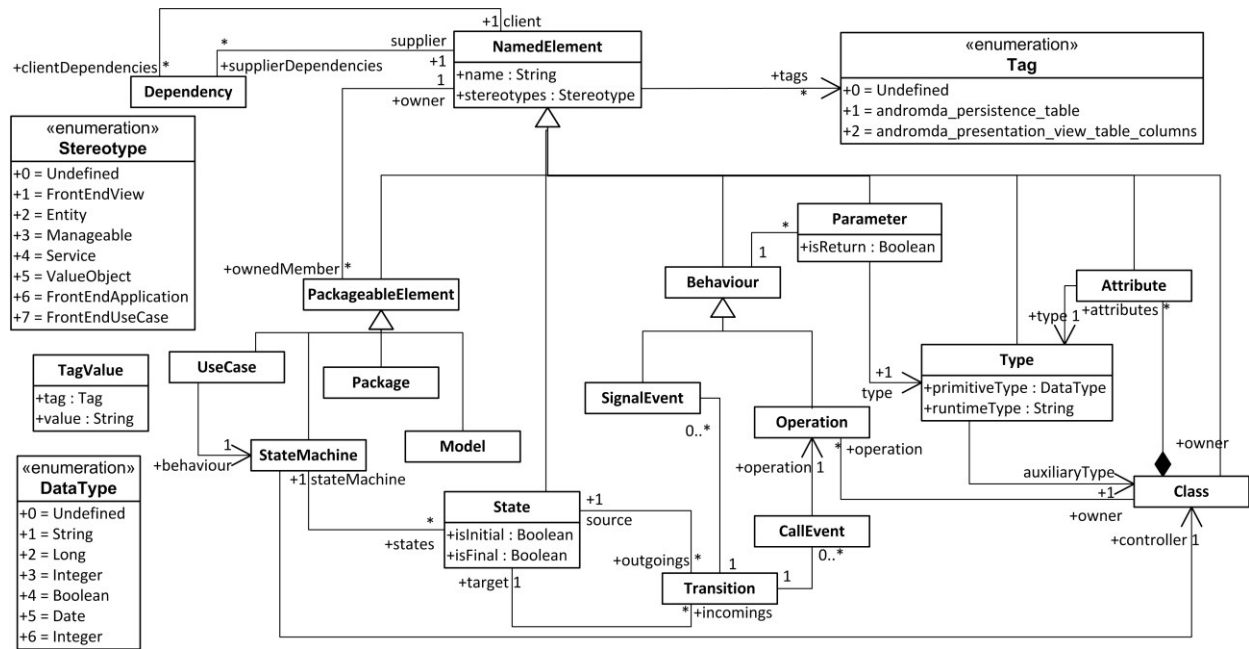


Figure 46 The Meta-Model of the AndroMDA-based Platform

Note that the meta-model in Figure 46 is only used for generating a model from which code can be generated using AndroMDA. It should not be considered as an official AndroMDA model. AndroMDA elements are uniquely identified as specified in Table 11.

Table 11. AndroMDA Elements Unique Keys

Element	Key Combination
NamedElement	Name, Owner (i.e. PackageableElement)
Model	Name
Transition	Source, Target
Use Case	Name
Operation	Name, Owner (i.e. Class)
Parameter	Name, Owner (i.e. Operation or SignalEvent)
SignalEvent	Name, Owner (i.e. Transition)

Top relations are as follows:

1. *application_model* is the main relation that check the APSM application, its use cases and data composites, and creates the application packages and their containing classes. A generated application contains the following four packages:

- i. *services*, where service classes are created and service operations are placed.
 - ii. *vo*, where data composites are mapped to AndroMDA value objects with the same attributes and associations
 - iii. *domain*, where data entities are mapped to AndroMDA entities.
 - iv. *web*, where every use case and its corresponding state machine and controller are modeled in a separate sub-package. This is the main source of other relations in this transformation.
2. *dependencyControllerService* is a relation that creates the required dependency relationships from controllers to services
 3. *presentation_frontEndView* is a relation that maps APSM presentation states to AndroMDA frontEndView states and handles other UI components and presentation details.
 4. *feedbackElement_addFeedbackUseCase* adds the required use cases to create the code of the feedback elements where required.
 5. *loginRequired_checkLoginState* adds the check for login information to the AndroMDA states needed.

The following functions are used to accomplish the following tasks that depend only on the target model *andromda*:

- *addStereotype* adds a stereotype to the collection of stereotypes of a named element.
- *addOutgoing* adds a transition to the collection of outgoing transitions of a state.
- *addIncoming* adds a transition to the collection of incoming transitions of a state.

7.1.2 A Sample APSM-to-AndroMDA Relation

The relation *dataComposite_valueObject* is listed in Listing 13 to Listing 16. This relation creates the *andromda* value object and services for every data composite found in the *apsm*.

Listing 13 The Relation *dataComposite_valueObject*: Declarations and Checkonly Domains

```
01:relation dataComposite_valueObject {
02:    dataName:String;
```

```

03:     compositeName:String;
04:     applicationName:String;
05:     serviceName:String;
06:     operationName:String;
07:
08:     checkonly domain aPSM application1:apsm::Application {
09:         name=applicationName
10:     };
11:
12:     checkonly domain aPSM dataComposite1:apsm::DataComposite {
13:         name=compositeName,
14:         service=service1:apsm::Service{
15:             name=serviceName,
16:             operations=serviceOperation1:apsm::Operation {
17:                 name=operationName
18:             }
19:         },
20:         dataEntities=entity1:apsm::DataEntity {
21:             name=dataName
22:         }
23:     };
24:

```

- Lines 2-6 declare the variables.
- Lines 8-10 check the *apsm* for an application.
- Lines 12-23 find the data composites within the *apsm*.
- Lines 14-19 discover the services associated with the *apsm* data composites.
- Lines 20-22 discover the data entities related to the *apsm* data composite.

Listing 14 The Relation *dataComposite_valueObject*: Enforcing the Value Objects.

```

25:enforce domain andromDA model2:andromda::Model {
26:    name='Data',
27:    ownedMember=package2:andromda::Package {
28:        owner=model2,
29:        name=applicationName,
30:        ownedMember=package2_1:andromda::Package {
31:            owner=package2,
32:            name='org.modewis.'+applicationName,
33:            ownedMember=voPackage2:andromda::Package {
34:                name='vo',
35:                owner=package2_1,
36:                ownedMember=valueObject2_1:andromda::Class {
37:                    name=compositeName.firstToUpper()+'VO',

```

```

38:     stereotypes=addStereotype(valueObject2_1, andromda::Stereotype::ValueObject),
39:         attributes=attribute2:andromda::Attribute {
40:             owner=valueObject2_1,
41:             name='id',
42:             type=andromda::DataTypes::Long
43:         },
44:         owner=voPackage2
45:     },
46: },

```

- Line 25 declares enforcing the *andromda* model.
- Lines 30-32 create the main package of the *andromda* application.
- Lines 33-37 create the *andromda* value objects correspondent to the *apsm* data composites.
- Line 38 adds the *ValueObject* stereotype to the created value object.
- Lines 39-43 create a default *id* attribute required to work with *MySQL*.

Listing 15 The Relation *dataComposite_valueObject*: Enforcing Services.

```

47:         ownedMember=package2_1_1:andromda::Package {
48:             name='services',
49:             owner=package2_1,
50:             ownedMember=serviceClass2:andromda::Class {
51:                 owner=package2_1_1,
52:                 name=serviceName,
53:                 stereotypes=addStereotype(serviceClass2, andromda::Stereotype::Service),
54:                 operation=operation2:andromda::Operation {
55:                     name=operationName,
56:                     owner=serviceClass2,
57:                     parameter=parameter2:andromda::Parameter {
58:                         owner=operation2,
59:                         name=compositeName.toLowerCase()+ 'VO',
60:                         type=type2:andromda::Type {
61:                             auxiliaryType=valueObject2_1
62:                         }
63:                     }
64:                 }
65:             }
66:         }
67:     }
68: }
69: };
70:

```

- Lines 47-52 create the package required to contain services and the service classes too.
- Line 53 adds the *Service* stereotype to the created value object.
- Lines 54-56 add the required operations.
- Lines 57-62 create the operation parameters.
- Lines 60-62 add a parameter to pass the data composite itself to the service operation.

Listing 16 The Retry Links from the Error State

```

71:     where {
72:         service_class(application1, service1, model2);
73:     }
74: }
```

- Finally, lines 71-73, map the application to service for further processing and adding more details.

7.1.3 A Sample AndroMDA Model

Figure 47 presents the *andromda* classes supporting the behaviour of the *Open Poll* use case presented in Chapter 6. Our AndroMDA meta-model suggests a structure very similar to the one of the APSM and hence the *andromda* models are typically composed of the same elements as their original *apsm*. For the most part the *APSM-to-AndroMDA* adds required stereotypes, tagged values and packages. For example, in Figure 46, the mapping will give the class *Election*, the stereotype *Entity* while the class *Poll* has the stereotype *Manageable* because it requires being updated in use case *Open Poll*.

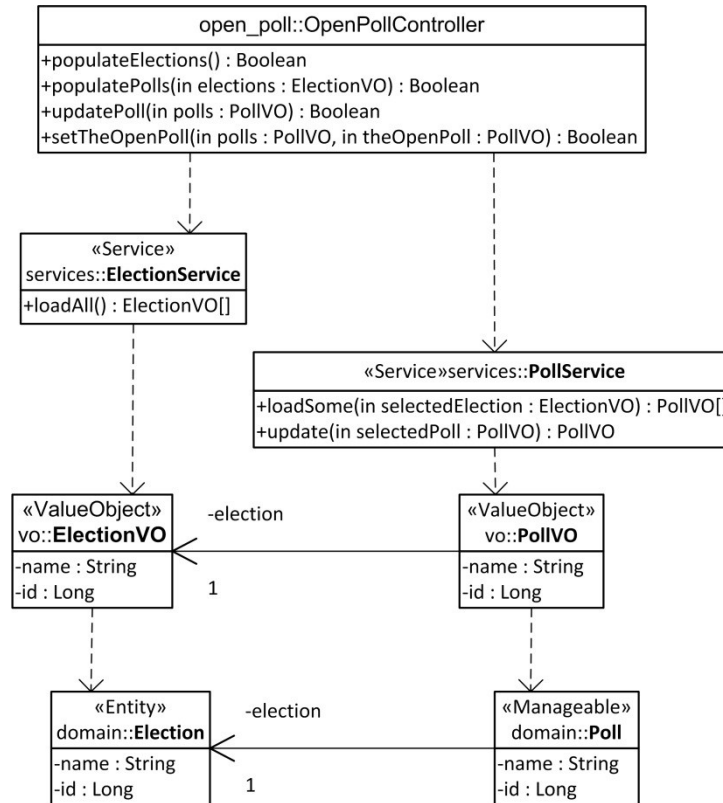


Figure 47 *Andromda* classes mapped from *apsm*

7.2 APSM-to-WebRatio

The suitability of our abstract web model for non-refining mappings was also examined. Non-refining mappings are mappings to non-UML models (Section 2.6.3). A good example of a non-UML, and yet very popular model is WebML. A set of mappings from APSM to a specific configuration of WebRatio – which is a WebML-based tool – have been designed. WebML supports a hierarchy of Site View, Area, and Sub-Areas or Pages recursively. Modeling units are either content units that are devoted to hypertext content or operation units that facilitate controlling and data-related operations. Pages are categorized as:

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

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

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

- a. For every site view, a default homepage is created named after the view name followed by the term 'Home'.
 - b. For every use case related to an actor
 - i. For every state machine of the use case.
 - 1. a page is created named after the first presentation state of the state machine,
 - 2. a link from the default homepage to the page created in previous step is added.
2. Entry Units: For every signal event found on a transition exiting a presentation state an entry unit is inserted within the page corresponding to the presentation state.
- a. For every data operation associated with the signal event, the entry unit is paired with an operation unit. The type of the operation unit depends on the type of the data operation. For example, a data create operation would be mapped to a create unit.
3. Operation Units: For every pair of a signal-event/call-event found in a state machine, the mapping of the signal event (i.e. the operation unit) needs to be supplied by the following:
- a. Fields that are created based on the call event parameters
 - b. An OK link that leads to a target page in case the operation is successful. The target page is the mapping of 1) another presentation state, 2) the homepage of the view or 3) the first presentation state of another state machine.
 - i. The first case is taken when the transition that exits the choice after the call event guarded with a true condition leads to a presentation state.
 - ii. The second case is selected if the transition ends at the final state.
 - iii. The third case is chosen when the transition ends at the final state and the final state is associated with another use case.

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

7.2.1 APSM-to-WebRatio QVT Relations

Figure 48 depicts an abstract model used as meta-model for WebRatio applications in the APSM-to-WebRatio transformation. Table 12 lists the keys used for uniquely identifying WebML elements in this transformation.

Table 12. WebML Elements Unique Keys

Element	Key Combination
WebProject	Name
WebModel	Name
SiteView	Name
Unit	Name, siteView
Entity	Name
Field	Name, Unit
Link	Name
Attribute	Name, Entity
Parameter	Name, Link

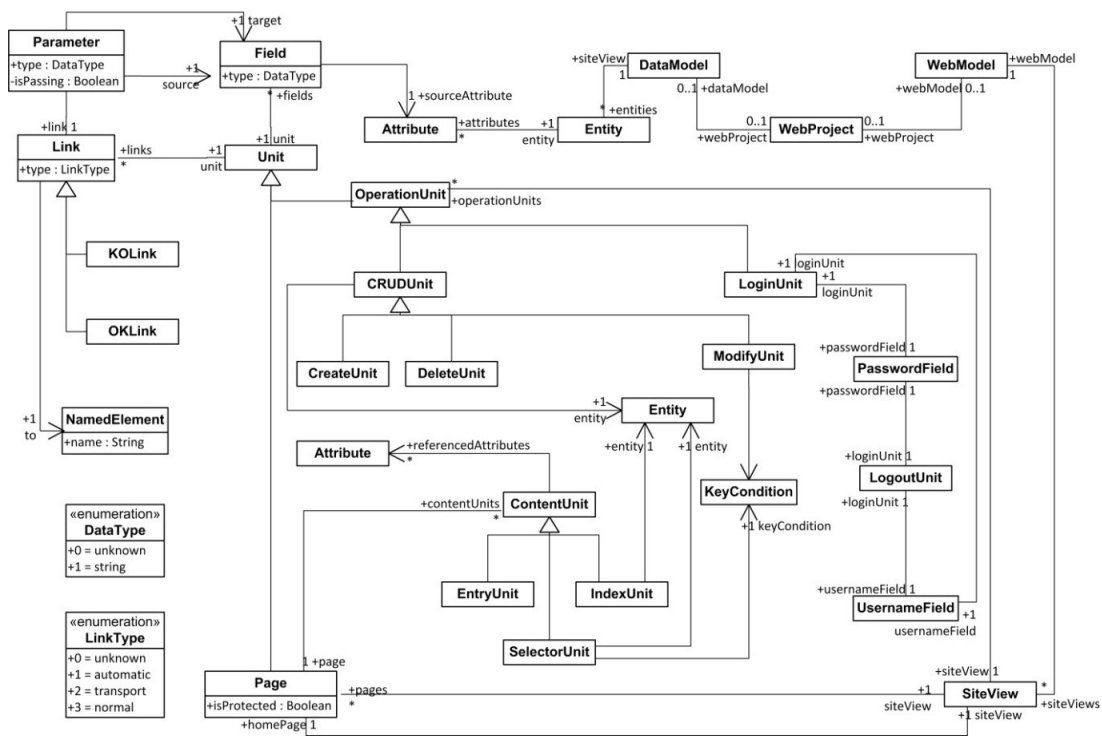


Figure 48 The Meta-Model of the WebRatio-based Platform

Top relations that validate the transformation are as follows:

1. *application_webProject* maps the *apsm* application to a web project, web model and a default site view.
2. *actor_siteView* maps every *apsm* actor to a site view and adds a default homepage to every site view.
3. *entryChoiceAtUpdate_linkUseCaseState* and other relations with the same title but different CRUD property map *apsm* transitions carrying call events to the required WebRatio link.
4. *entryChoiceAtCreate_linkUpdate*, *entryChoiceAtCreate_linkCreate* and other relations with the same title but different CRUD properties map *apsm* transitions carrying call events to the required operation units. These relations complete the task of the relations of item 3.
5. *signalEventTransition_linkCreateUnit*, *signalEventTransition_linkUpdateUnit* and other relations with the same title but different CRUD properties map the APSM transitions carrying signal events to the relevant operation units. The difference between these relations and the items 3 and 4 is that these relations map to links outgoing from the presentation unit while the relations of item 3 and 4 map to the target operation unit and their outgoing links.
6. *updateCallEvent_updateUnit* and other CRUD relations map the details of CRUD operations.
7. *startup_homePage* relation maps the startup use case to an appropriate homepage.

The following functions are used for the transformation:

1. *getWebRatioType* returns WebRatio primary field, attribute and parameter types per input *apsm* types
2. *findTargetFor* finds a target field for a parameter based on simple name matching. Reader may note that this is the same as the default mechanism WebRatio suggests.
3. *findIDSourceField* finds the id field.
4. *addPage* adds a page to the collection of pages of a site view.

5. *addAttribute* adds an attribute to the collection of attributes of an entity.
6. *getSourceAttribute* finds a source attribute for a CRUD operation parameter based on simple name matching. Note that this is the default mechanism suggested by WebRatio.

Figure 49 The Object Diagram of the *webratio* Model of *Open Poll* Use Case

Figure 49 shows the object diagram of the *webratio* model of the use case *Open Poll*. A site view object is created for the admin. It has two pages, *AdminHome* and *OpenPoll*. The former is created as a default page for every site view and the latter corresponds to the presentation state of

Open Poll use case. The *OpenPoll* page has a main entry unit *open*. This unit relates to the operation and CRUD units through links. The first link has the Select Unit *loadAllElections* as its source. Note that *webratio* does not have source and target for links as in most other metamodels. The source of a link is the unit that has the link as a member in its *links* collection. For example, *loadAllElections* unit has *loadAllElections_loadSomePolls* and *loadAllElections_ErrorPage* in its *links* collection. This means *loadAllElections* is the source of *loadAllElections_loadSomePolls* and *loadAllElections_ErrorPage* links. The target is specified using the *unit* property of the link. Links handling errors are recognized as *KOLink* as required by *webratio*. All *KOLinks* of a use case are directed to the same *ErrorPage*. This *ErrorPage* has an *ErrorMessage* field, which will be customized based on the unit that caused the error.

7.3 APSM-to-GWT

Google Web Toolkit (GWT) is a development framework mainly for client-side design of Web 2.0 applications using an asynchronous service-based architecture. GWT extends Java and supports the design and installation of UI components. The GWT itself does not support the Java Database Connectivity (JDBC) [JDBC2010] but it is possible to use JDBC to connect an RDBMS through an extension. The platform chosen in this thesis uses the Java Data Objects (JDO) [JDO2010] mechanism to support data-related features of the Google Application Engine (GAE) [GAE2011]. The application is deployable on the Google web server. A typical GWT application has a server and client package as well as a shared package. The client package includes the definition of the page UI components while the server side implements the services. Our configuration allows one service for each data composite. Although it is possible to develop Web 2.0 applications using the other platforms defined in this thesis, the mapping to GWT provides a more convincing evaluation regarding Web 2.0 because of its built-in Ajax capabilities.

Figure 50 shows a high-level visualization of the mapping in this section. Classes from package *modewis* are mapped to classes from package *gwt*; an association between two such classes is read as *maps to*. A panel is assigned to every actor. This supports the definition of different views for different actors. All other pages and forms are defined as sub-panels of these panels. The following is a summary of other mappings in Figure 49:

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

Figure 51 shows the model used as the meta-model in the APSM-to-GWT transformation.

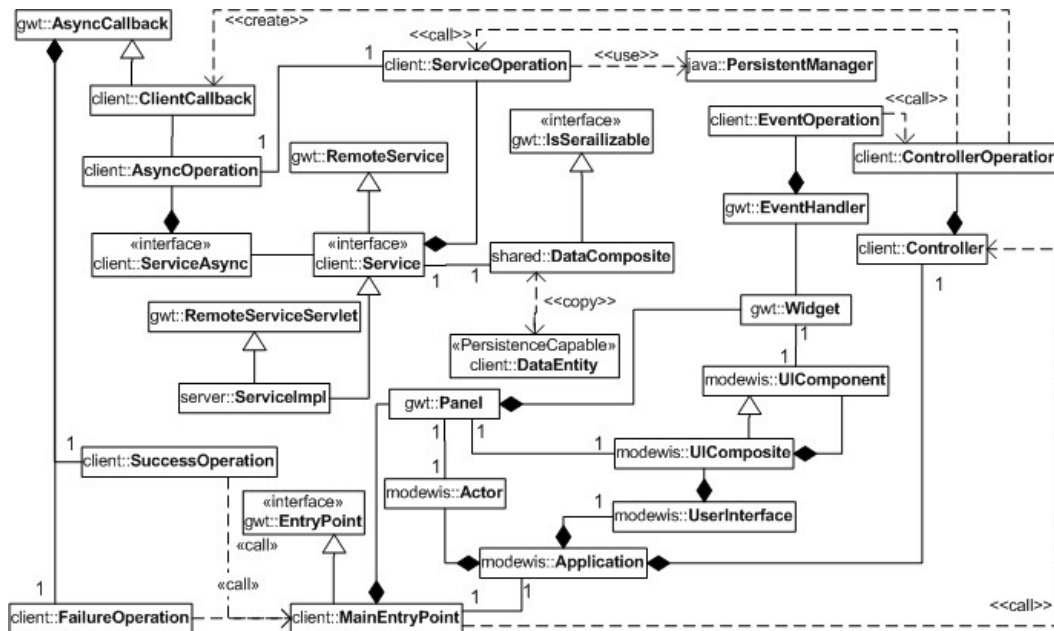


Figure 50 APSM-GWT Mapping: A high-level viewpoint

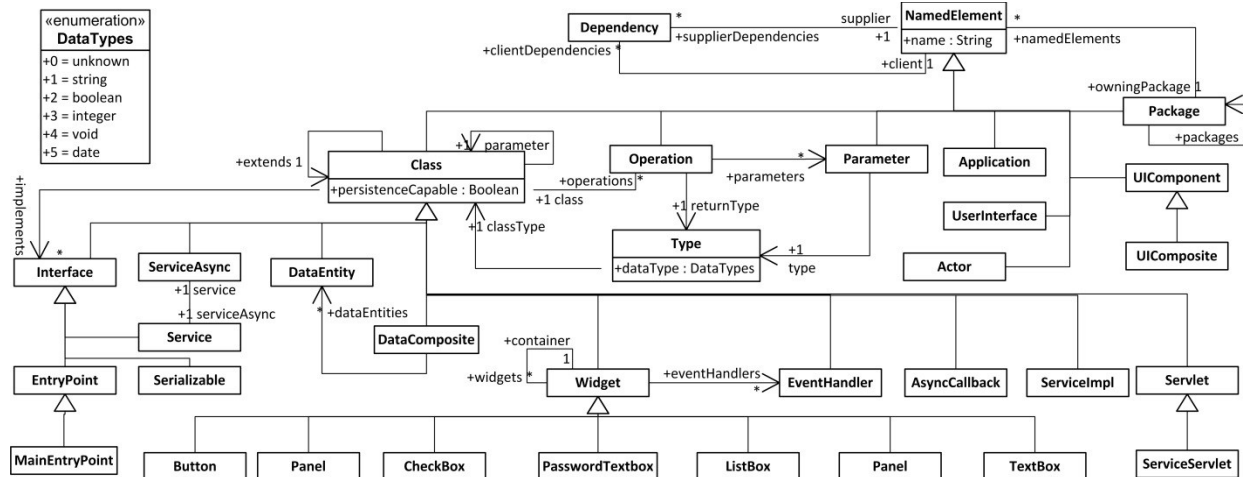


Figure 51 The Meta-Model of the GWT-based Platform

Table 13. GWT Elements Unique Keys

Element	Key Combination
Widget	Name, Container
Operation	Name, Class
Package	Name
Application	Name
Class	Name, OwningPackage

Top relations in this transformation are as follows:

1. *application_application* maps the APSM application to a GWT application and creates the required package structure.
2. *usecase_package* maps each APSM use case to a GWT package within the package *client* and creates the controller, default UI elements and required operation calls and event handlers.
3. *service_service* maps each APSM service to the relevant GWT service, asynchronous services, callback operations and server implementations. This relation also maps the parameter and return types to GWT types whether they are primary types or class types.
4. *dataEntity_domainClass* maps APSM data entities to GWT classes within the package *domain* and creates pertinent attributes and associations.

5. *presentationState_clientPackage* maps the APSM presentation and navigation structure throughout the GWT packages, panels and widgets.

The function *getGWTPrimitiveType* is used to obtain the appropriate GWT type based on the found APSM type for each parameter or attribute.

Figure 52 shows an object diagram of the *Open Poll* use case when mapped to *GWT*. According to this figure, a panel *OpenPollPanel* is created for the *Open Poll* use case. This panel has a controller *OpenPollController*. This controller has four operations. Except for the *setTheOpenPoll* operation, the three other operations call service operations. Two services *PollService* and *ElectionService* are provided to support the CRUD operations regarding *PollComposite* and *ElectionComposite*. Every service has an interface and an implementation. The service and its interface belong to the package *client* while the implementation belongs to the package *server*.

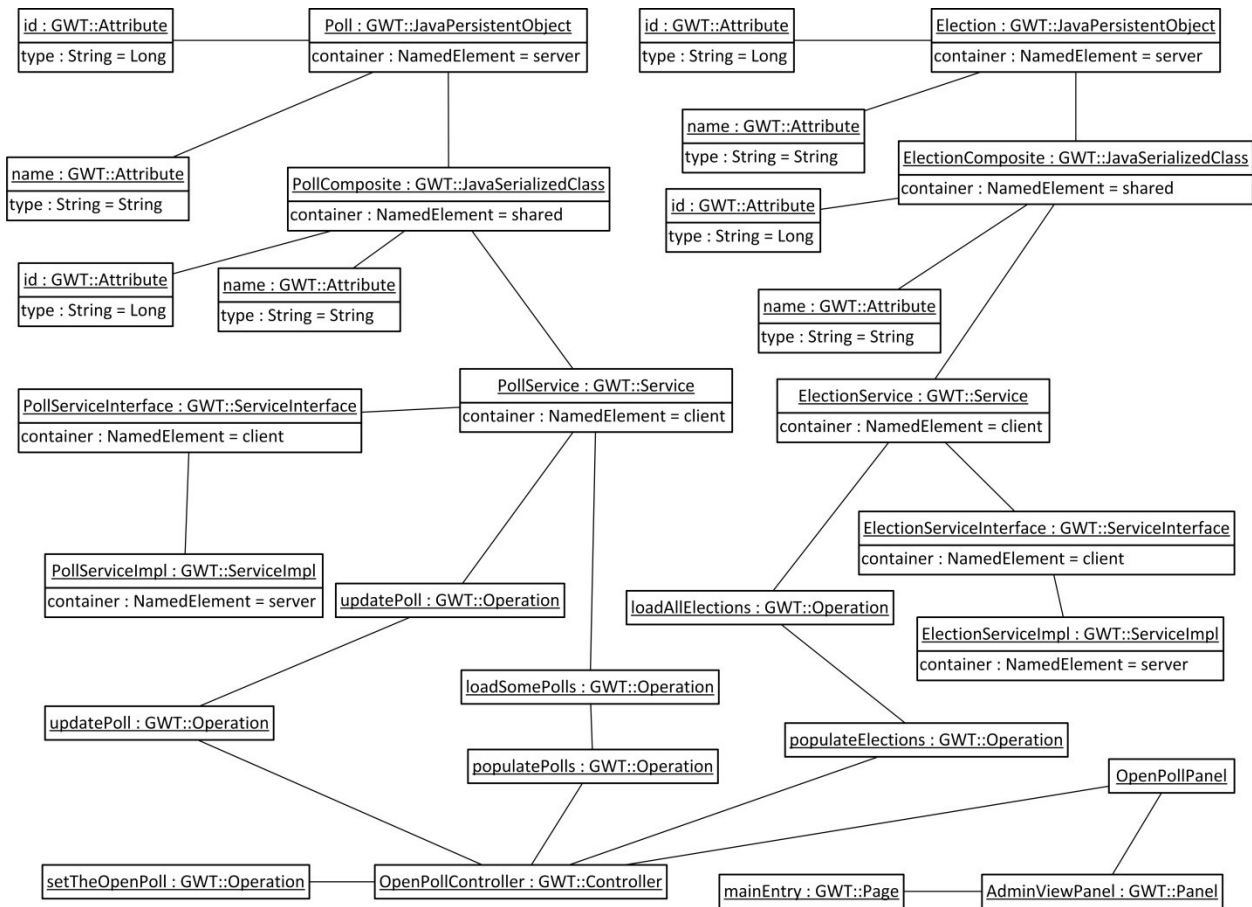


Figure 52 The Object Diagram of *Open Poll* use case in *GWT*

7.4 APSM-to-Net

Applications defined based on this platform are developed using ASP as scripting language, C# as programming language, SQL Server as the database management system and .Net as the application framework. A specific feature of this platform is the separation of *DataService* and *DataAccess*. This adds a new level between the application and the database when compared to the other specific platforms used in this thesis. The class *DataAccess* facilitates access to the relational SQL Server database and performs the required data conversion. The class *DataService*, in turn, handles calls to data access and returns data elements to controllers. A specific class *GenericDataAccess* verifies data access requests and assigns them to the corresponding stored procedures. Thus, among all our specific platforms, our .Net-specific platform is the most loosely-coupled. Figure 53 shows a meta-model based on this platform. Table 14 lists the keys for elements of this meta-model.

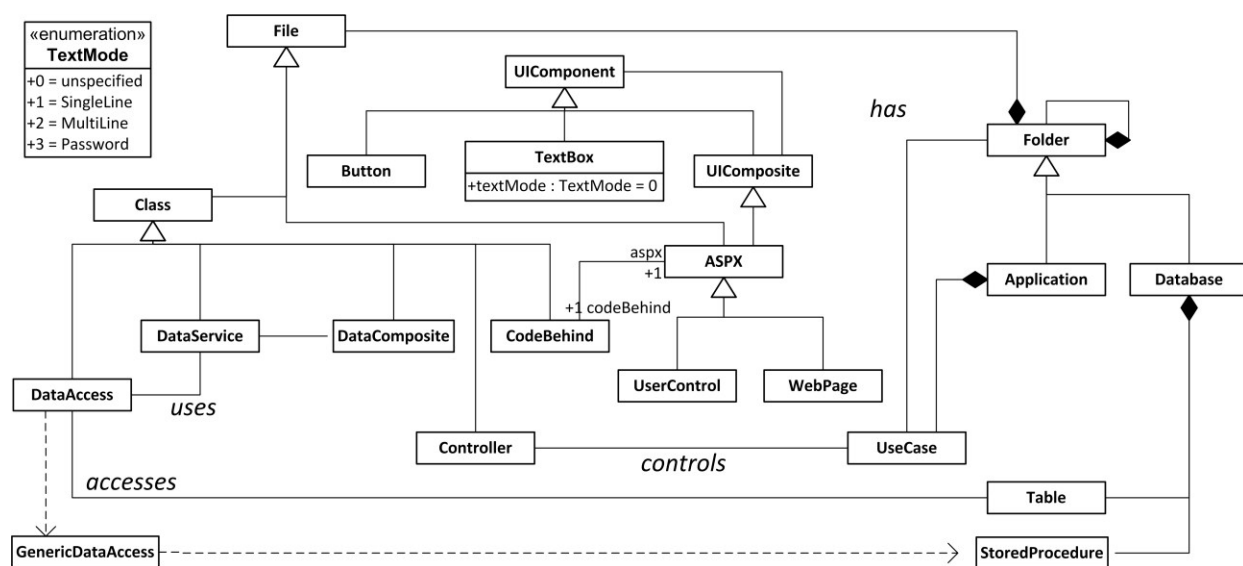


Figure 53 The Meta-Model of the .Net-based Platform

Table 14. .Net Elements Unique Keys

Element	Key Combination
Application	Name
Database	Name
Operation	Name, OwningElement
StoredProcedure	Name
NamedElement	Name, OwningFolder
UseCase	Name

Top relations in this transformation are as follows:

1. *application_application* maps the APSM application to a .Net application and creates the required folder structure. This also creates controllers for every use case.
2. *nextStep_nextStep* maps every use case attached to a final state to the same structure in the .Net platform.
3. *operationTrigger_webForm* maps APSM operation triggers and their enclosed UI components to a .Net user control and attaches the form on a web page. The web page is a mapping from the equivalent APSM presentation state.
4. *serviceCall_accessCall* maps an APSM call to a data service to the required calls to .Net data services as well as calls to .Net data access.
5. *presentation_userControl* maps the APSM presentation to an equivalent .Net control on a web page. This relation requires other relations to map APSM UI components to their equivalent .Net types such as textbox or button.
6. *dataComposite_dataComposite* maps every APSM data composite to the required C# classes. These classes are required for the data composite itself, its required data services, the data access and the stored procedure pertinent to the query types the data composite is involved with.

Also, the function *getQueryType* is used to obtain the appropriate .Net query type based on the found APSM query type for each operation.

Figure 54 shows an object diagram of the use case *Open Poll* in .Net platform. A user control is created for the use case *Open Poll*. Both for *Poll* and *Election*, a table is created in the *EMS* database. Required data service operations are mapped to stored procedures. There are three stored procedures *updatePoll*, *selectSomePolls* and *loadAllPolls*. Stored procedures are accessed by *EMSGenericDataAccess*. The *EMSGenericDataAccess* is a unified gate for all data access classes, such as *ElectionAccess* and *PollAccess* to call stored procedures. Data access classes are in turn called by data services, which transform data composites to the format required by the relational tables.

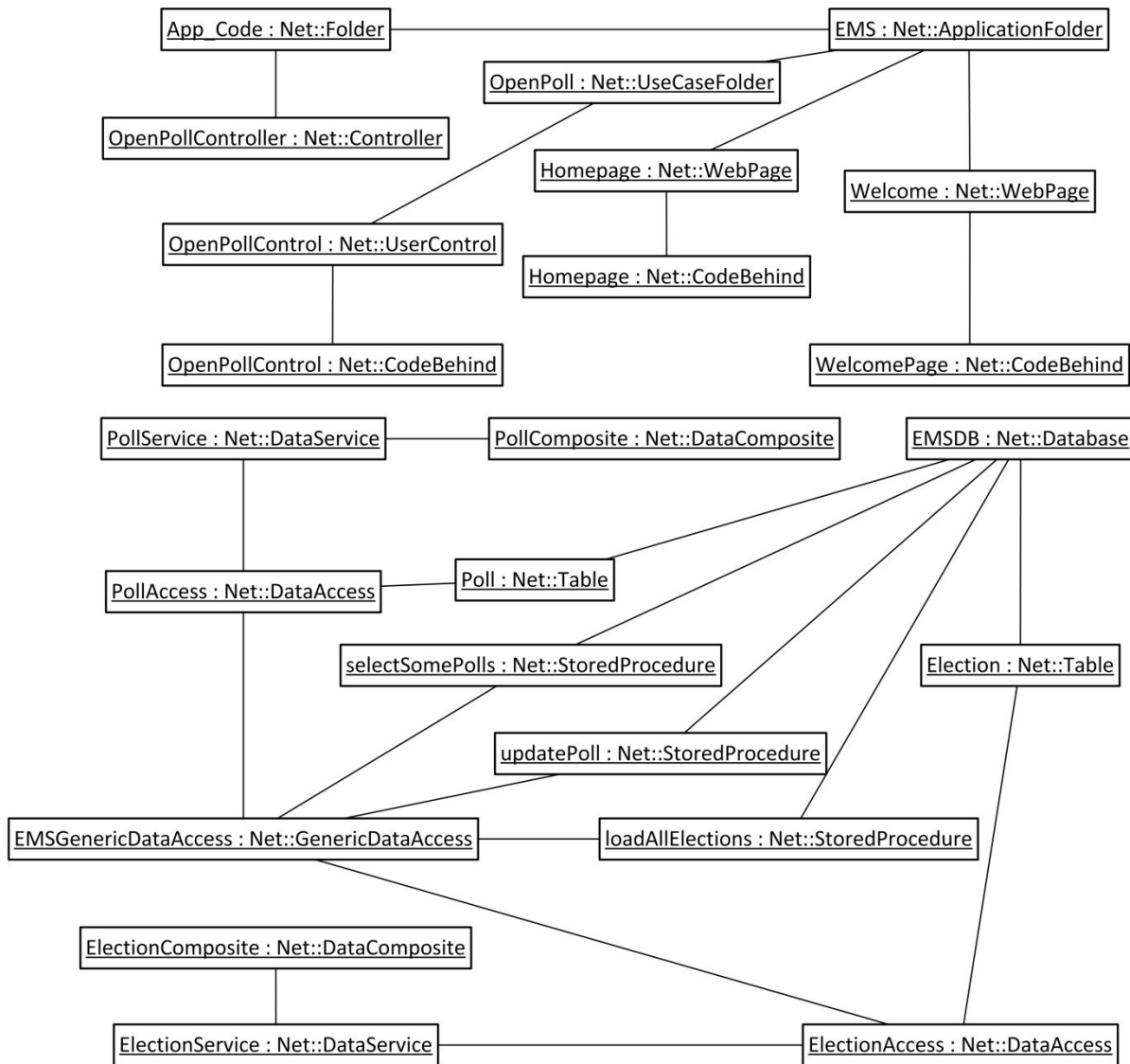


Figure 54 An Object Diagram of the Open Poll Use Case in .Net Platform

7.5 Summary

This chapter covered the implementation of the mappings that are supplements to the APSM. These transformations are used to map the APSM to four specific platforms. These platforms are based on different technologies: Google, Microsoft .Net, WebRatio (WebML), and AndroMDA (J2EE). In order to specify the platform-specific transformations, meta-models of the target platforms were also defined. These models were presented in Figure 46, Figure 48, Figure 51 and Figure 53. These meta-models do not intend to be thorough meta-models of the target platforms. They contain only the details required to build the *spsm* applications as mandated by

the *spsm* transformations. A full implementation of the transformations in this chapter can be found in [MDWSQVT2011].

Chapter 8. IMPLEMENTATION: A VISUAL OVERVIEW

In this chapter, we introduce a development approach based on the APSM and a set of utility tools supporting that approach. We use examples selected from an Election Management System (EMS) case study. We first review the implementation. Then, we present a brief introduction to the EMS followed by a presentation of the visual models created for the *pim* of the EMS. This visual *pim* is used as input to the approach. Afterward, snapshots and models pertaining to utilities used to create and help investigate the *pim* and the *apsm* from the visual input are provided. Finally, we present snapshots of the EMS models built for specific platforms.

8.1 The Implementation Suite

The transformations discussed in Chapter 6 and Chapter 7 were implemented using MediniQVT [Medin2008]. The Eclipse Modeling Framework [EMF2009] was used to generate the meta-models in the form of ECORE models. We used an open-source diagramming tool *yEd* [YED2010] to define our graphical input models. These input models are then transformed to EMF using a utility that we developed. Once the transformations are executed in MediniQVT, the output EMFs may be transformed to specific platforms using other utilities that perform file conversion transformations. The implemented approach is named MODEWIS, which stands for Model-Driven Development of Web Information Systems. Figure 55 shows how MODEWIS interacts with other components in this implementation.

MODEWIS components (Figure 55) are integrated using a number of utilities. A utility, *MODEWIS_PIM_Generator* transforms the *yEd* files to EMF [PIM2010]. Once the transformations are executed in MediniQVT, the output EMFs may be transformed to specific platforms using other utilities that perform file conversion transformations such as *MODEWIS_GWT_Generator* [MDWSGWT2010]. A VBA module, *MODEWIS_ProofRead* [MDWSPR2010] has been implemented under Microsoft Visio [Vis2010] to visualise the results of transformations and help verify their correctness and validity. A typical scenario for working with MODEWIS – e.g. targeting GWT - is as follows:

- 1 Create the *pim* according to the meta-model of Figure 28 using *yEd* [YED2010]. A set of *yEd* image files for defining the *pim* using the visual language of Section 5.1 is available online [WKSH2010].
- 2 Transform the *yEd* file to EMF using *MODEWIS_PIM_Generator* [PIM2010].
- 3 Map the resultant *pim* to *apsm* using *PIM_APSM* QVT relations [Fatol2010-1].
- 4 (Optional) Investigate the created *apsm* using the *apsm* tab of *MODEWIS_ProofRead* [MDWSPR2010].
- 5 Map the resultant *apsm* to *gwt* using *APSM_GWT* QVT relations [Fatol2010-1].
- 6 (Optional) Investigate the created *gwt* model using the GWT tab of *MODEWIS_ProofRead* [MDWSPR2010].
- 7 Transform *gwt* to code using *MODEWIS_GWT_Generator* [MDWSGWT2010].

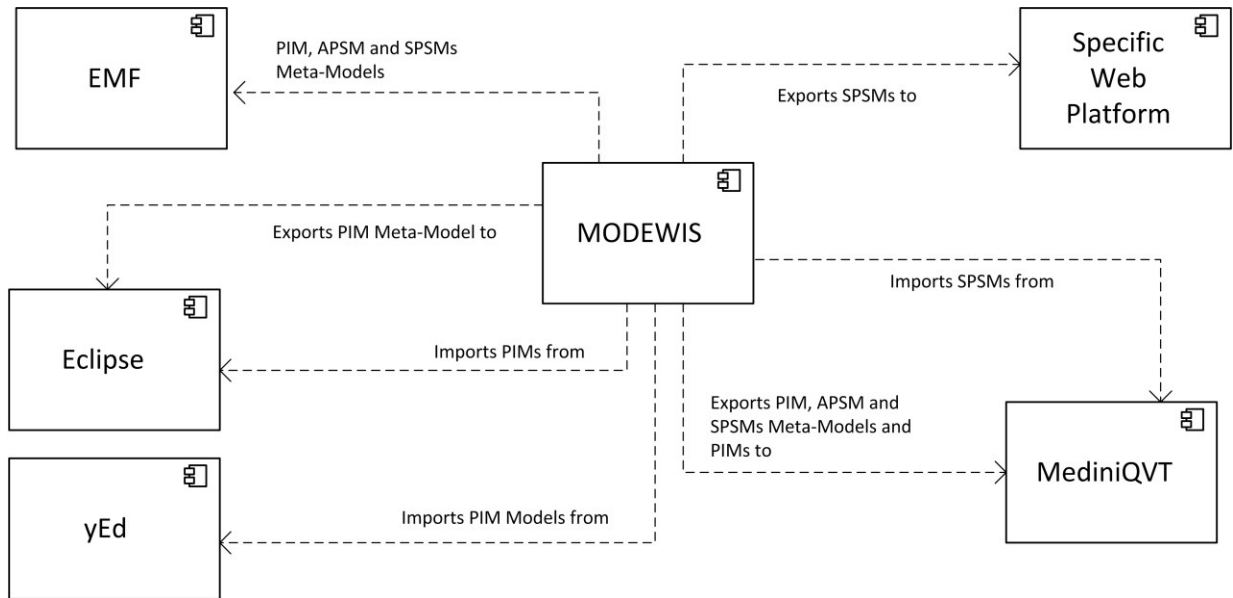


Figure 55 The Architecture of MODEWIS

8.2 The Election Management System

We use an example from a case study based on an Election Management System (EMS) [Leth2001]. The intent of the EMS is to manage information regarding elections and polls. The system provides facilities for data-related activities and management of the voting process. It also provides some information to journalists. Figure 56 presents the use case diagram of the

EMS. A full implementation of EMS would encompass more use cases. For example, we have not included use cases required for deleting or updating information elements.

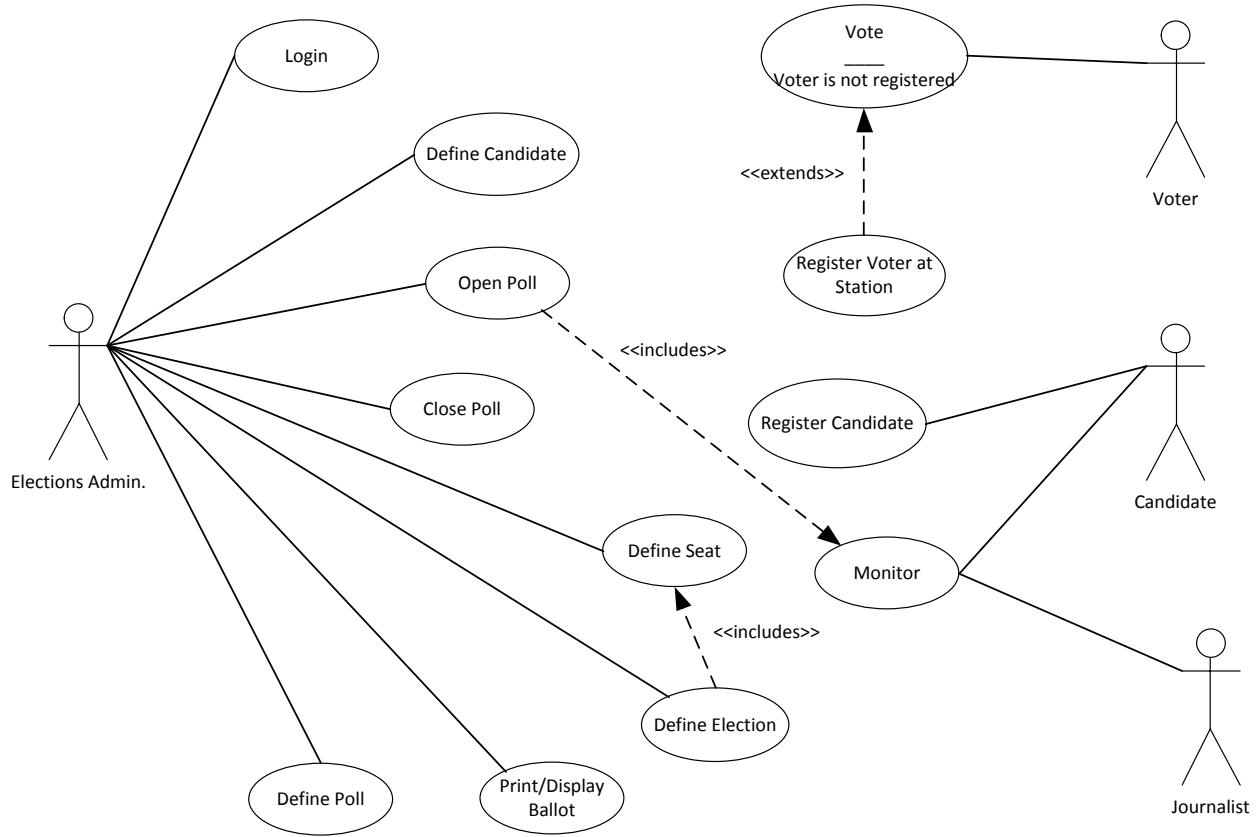


Figure 56 Use Cases of the EMS

8.3 Visual PIMs

The design of a *pim* is a process that involves a collection of symbols [WKSH2010]. In our current implementation, we use the visual design tool yEd [YED2010]. In this section, we review three use cases, *Login*, *Open Poll* and *Vote*. Figure 57 shows the visual input created for the use case *Login*.

According to Figure 57, the state *Login* has a presentation, which is included in an upper-hand presentation named *Welcome*. The presentation *Login* includes another composition named *Login*, which is associated with submit button *Login*. Grouping this button with other components in a single group named after the button means that these other components are used in the form submitted by the button. The form has two components; *Username* is an input field and *Password* that is a password field. The button *Login* performs two operations: one query operation over the data element *User* and an *Add* operation over the page variable *loggedUser*.

The arrow from the query operation to the add operation denotes that the add operation needs the output of the query operation as an input. Hence, the operations are automatically ordered as query done first and add performed next. The query operation denoted using the symbol ? is a specific query, which retrieves only the single data item matching the input, if more than one data item matches this requirement, the first one will be picked; this query is named *selectOne*. The add operation, when performed on page variables is interpreted as a *setValue* operation, which assigns the value found at the source of the incoming arrow to the component found at the target of the outgoing arrow.

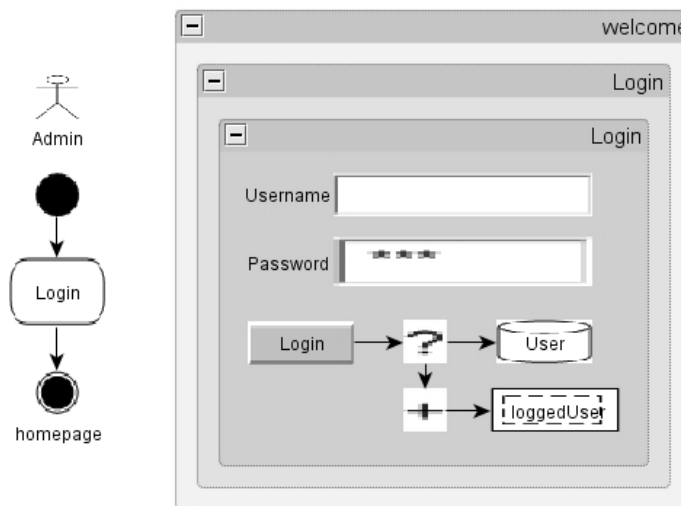


Figure 57 EMS Use Case, *Login*, *pim*

Figure 58 shows the *pim* of the use case, *Open Poll*. In this figure the presentation of the state *Open Poll* is contained within an upper-hand presentation *homepage*. The presentation *Open Poll* contains two other presentations, each of which includes a button. The name of each button matches one of the transitions outgoing from the state *Open Poll*. These transitions end at the final states annotated with the name of other state machines, *voting* and *welcome*.

8.4 Generating PIM EMF

Once the *pim* is visually designed, the next step is to use the PIM generator [PIM2010] to convert the visual design to an EMF file readable by Eclipse-based tools. The *PIMGenerator* is a part of the Java utility *MODEWIS* responsible for generating EMF from a visual *pim*. *PIMGenerator* first asks for the *yEd* files that contain the visual descriptions of use cases as shown in Figure 59.

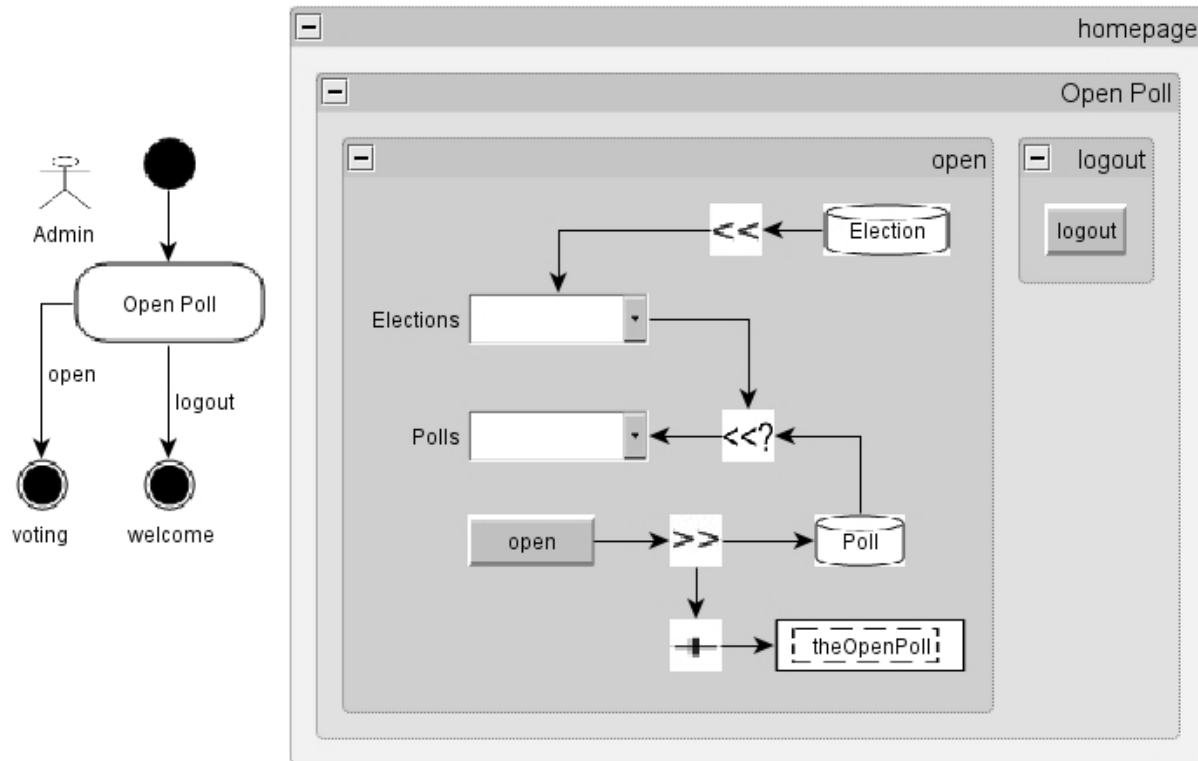


Figure 58 EMS Use Case, *Open Poll*, *pim*

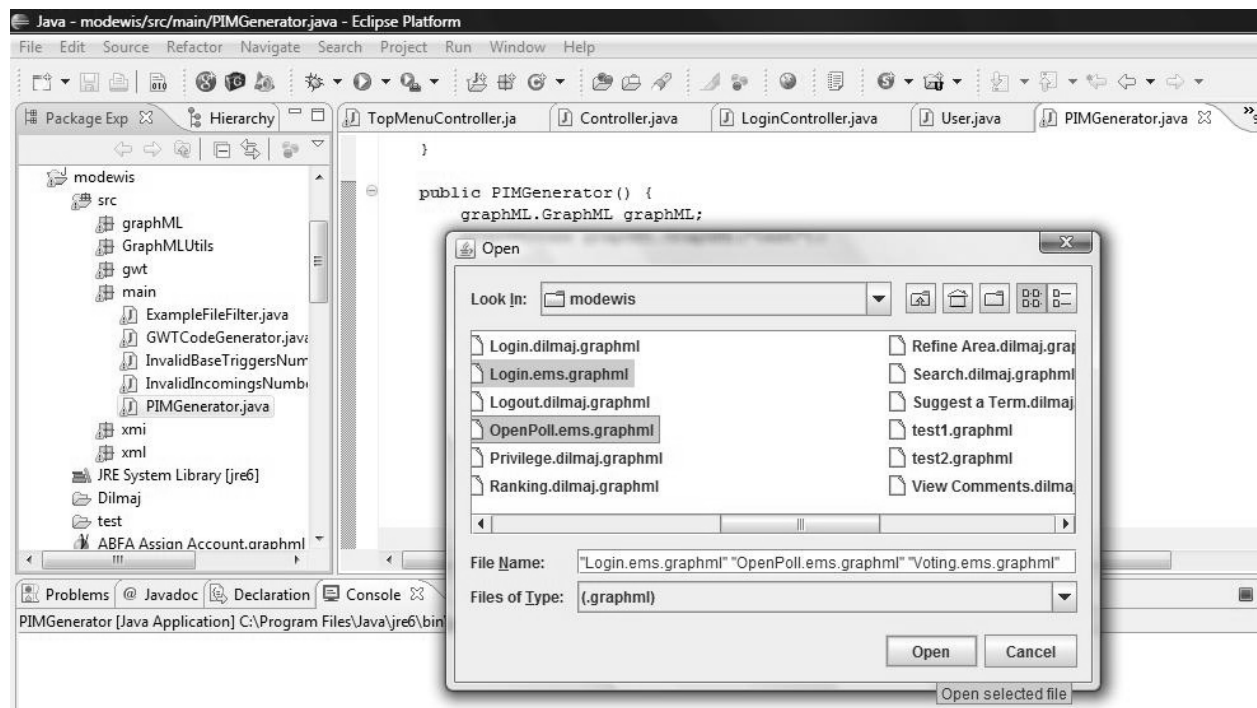


Figure 59 PIMGenerator - Selecting Visual PIM files

MODEWIS has a set of classes required to describe a *pim* created by *yEd* as shown in Figure 60. Once the *pim* files are read, the *addStateMachine* method of the class *GraphML* is called repeatedly to create the model of each state machine. This method first reads the *yEd* file and assigns every element an ID and a type. The type is determined based on the symbol associated with every item and the ID is directly chosen from the *yEd* file. Afterward, different group processing methods are called to process groups representing presentations, tables and operation triggers. Finally, elements are associated to their counterparts as required by the PIM meta-model. This includes assigning presentations to their corresponding states as well as assigning operation triggers to the transitions representing their action. Finally, operations found within each group are ordered properly based on their requirements and precedence as specified in Section 5.1.3. The generated model is then transformed to an XML-like structure to be saved in an EMF file. A snapshot of the EMF of the *pim* automatically created from the visual *pims* of Figure 57 and Figure 58 is shown in Figure 61, the complete listing is presented in Appendix D.

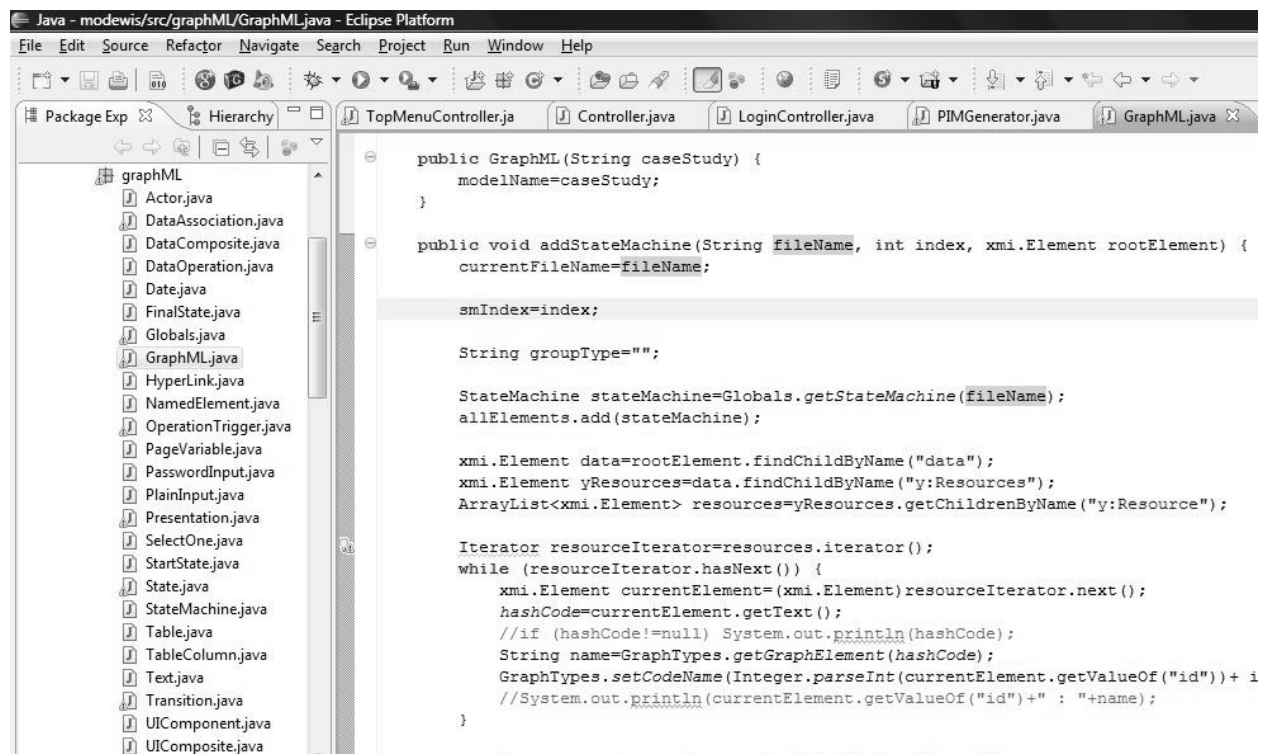


Figure 60 Classes that contain the definition of the PIM

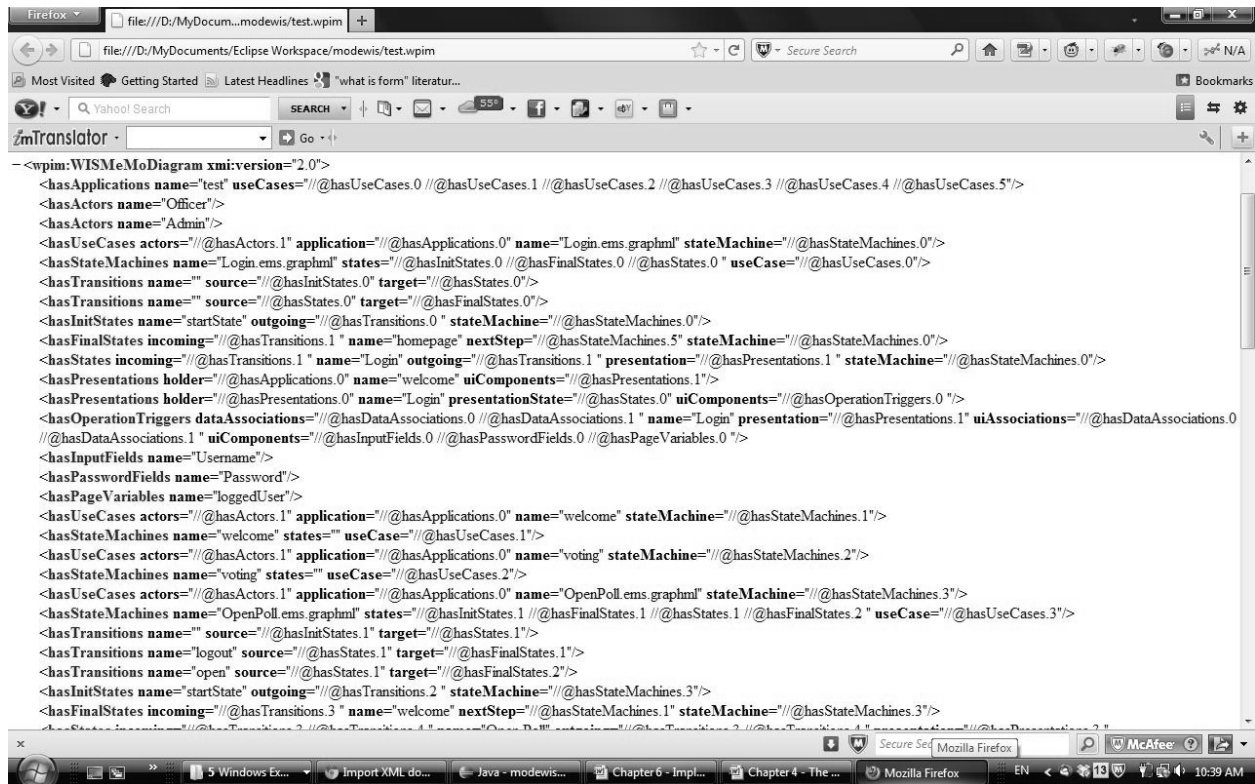


Figure 61 A Snapshot of the EMF file created by the PIM Generator

The model in Figure 61 is created in accordance with the EMF model of the PIM. This model is maintained using Eclipse and is designed based on the PIM defined in Section 5.1.5. Figure 62 depicts a snapshot of the Eclipse showing the *pim*.

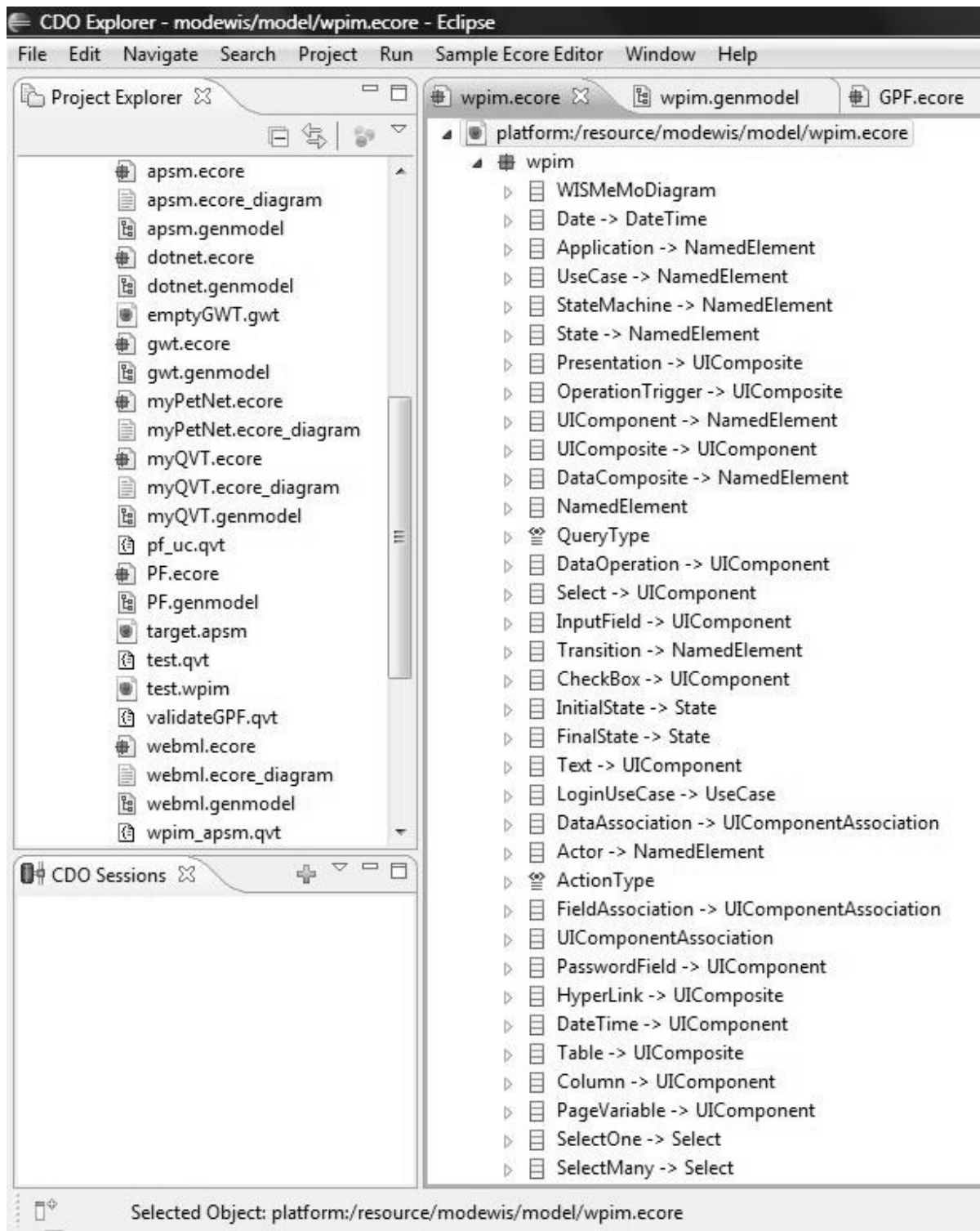


Figure 62 The PIM in Eclipse Environment

8.5 Mapping to APSM

The next step is to develop the *apsm*. The APSM is also implemented as an EMF model in Eclipse. Figure 63 shows the APSM as maintained in the Eclipse environment.

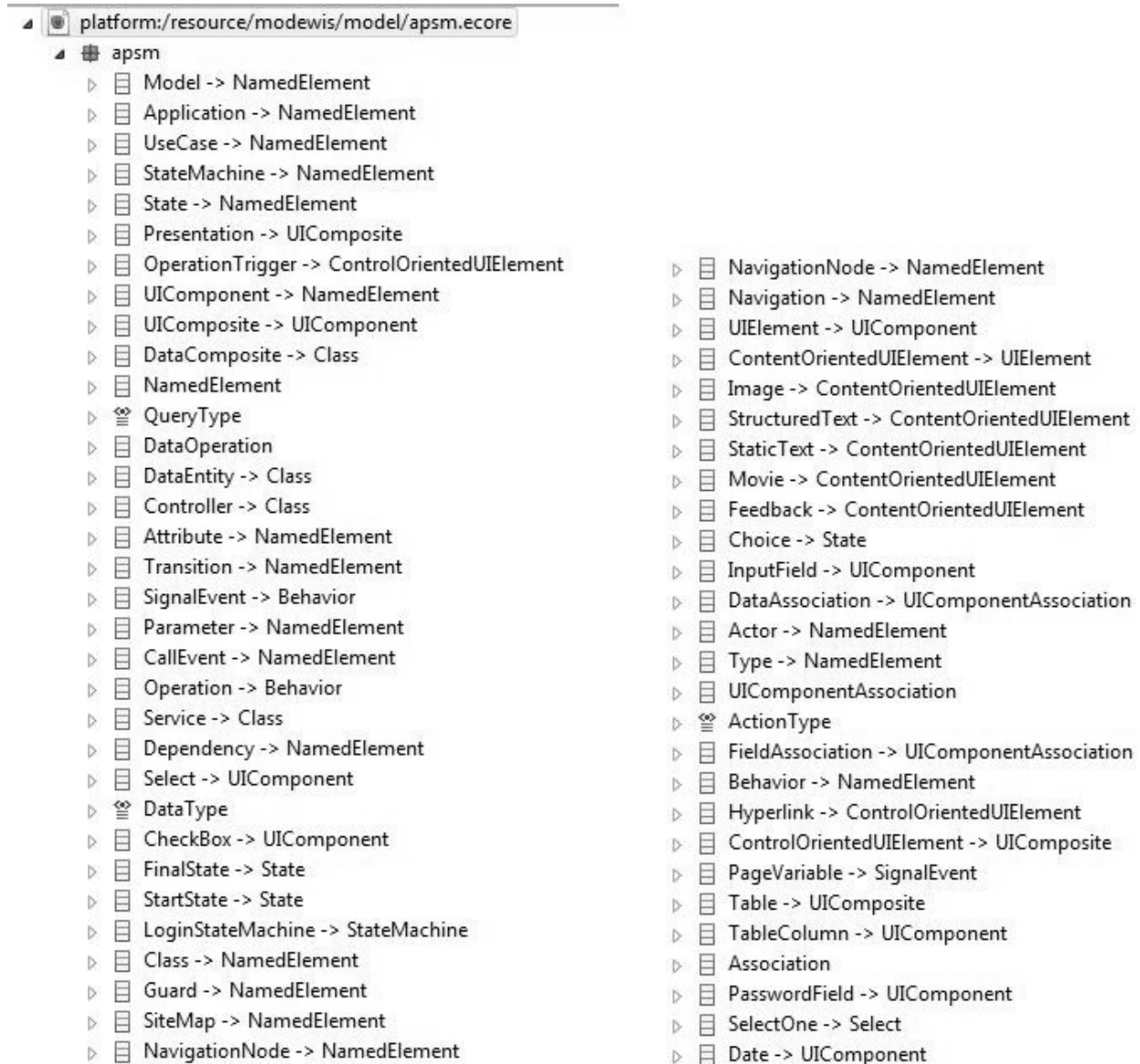


Figure 63 The APSM in Eclipse

Transforming a *pim* to an *apsm* is performed using the QVT relations implemented under MediniQVT. MediniQVT can run either as a standalone application or as an Eclipse plugin. Meta-models must be added through general Eclipse preferences as shown in Figure 64. Figure 65 shows the run configuration of pim-to-apsm transformations. The details of transformations are discussed in Chapter 5 and the complete listings are found in [MDWSQVT2011].

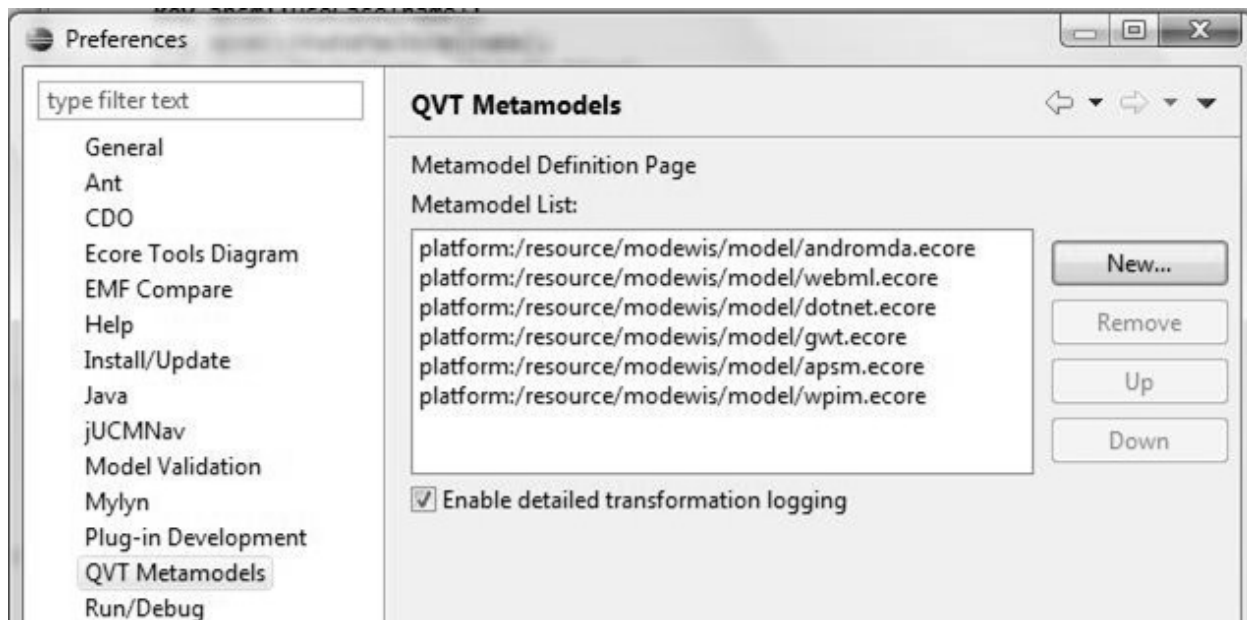


Figure 64 Defining Meta-Models for MediniQVT in Eclipse

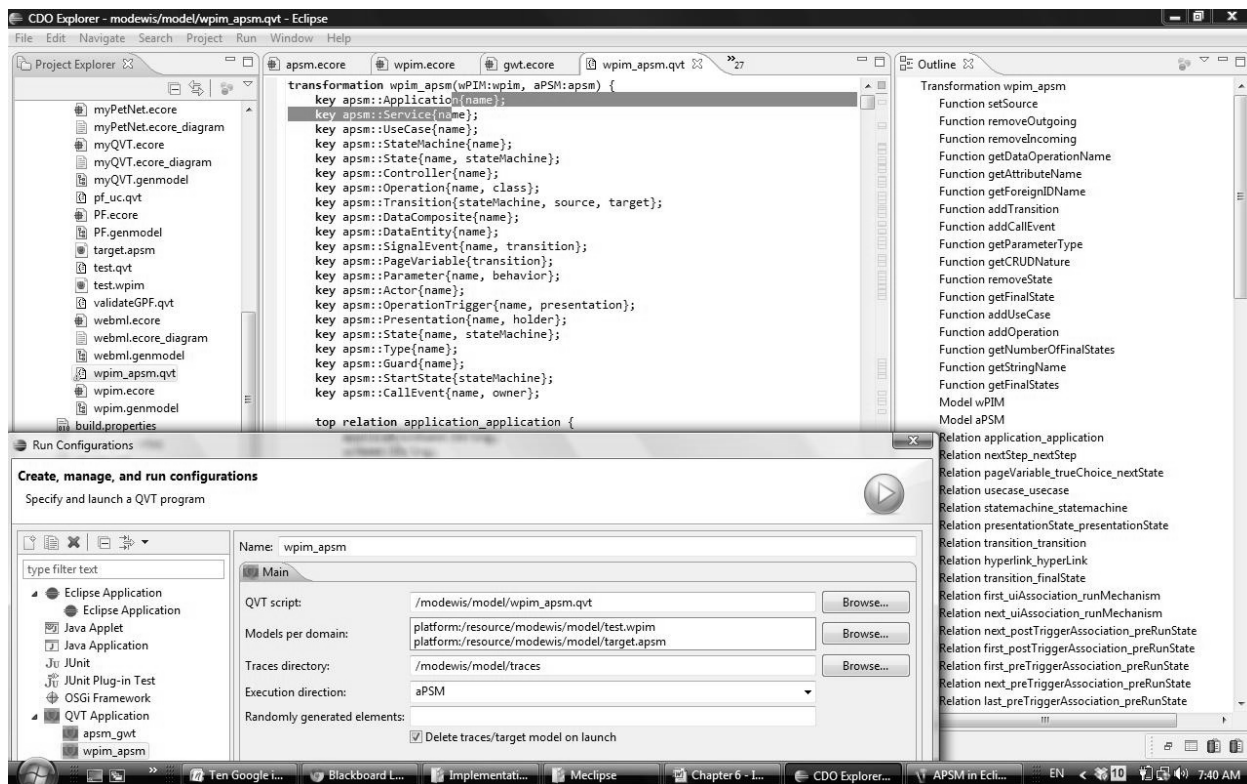


Figure 65 Run Configurations of pim-to-apsm in Eclipse

Figure 66 shows a snapshot of the created *apsm*, a complete listing is found in Appendix D. This *apsm* can be directly used to create a *spsm* or be investigated using the proofreading tool.

The proofreading tool is a Microsoft Visio file containing a few pages and VBA modules. Every module reads one of the outputs produced by our approach and creates a visual representation of the model. Figure 67 shows a list of these modules and pages in a typical view of the proofread. Figure 68 is a snapshot the *createAPSMStateMachine* macro in Visio, which creates the proofreading results for the state machines in an *apsm*; the complete file is available in [MDWSPR2010]. Currently, the output of this module needs to be laid-out manually. Figure 69 shows a snapshot of the state machine of the open poll use case in proofread environment.

```

- <xmi:XMI xmi:version="2.0" xsi:schemaLocation="http://apsm.modewis apsm.ecore">
  <apsm:Application name="test" useCases="/138 /173 /175 /109 /46 /180"/>
    <apsm:Presentation name="Vote" holder="/0"/>
      <apsm:Presentation name="Show Ballot" holder="/1" presentationState="/21" uiComponents="/3"/>
        <apsm:OperationTrigger name="Print" presentation="/2"/>
          <apsm:Presentation name="Enter Voter" holder="/1" presentationState="/22" uiComponents="/5"/>
            <apsm:OperationTrigger name="Login" uiComponents="/6 /7 /8 /9" presentation="/4"/>
              <apsm:InputField name="lastname"/>
              <apsm:InputField name="firstname"/>
              <apsm:Date name="birthdate"/>
              <apsm>PasswordField name="pin"/>
              <apsm:Presentation name="homepage" holder="/0"/>
              <apsm:Presentation name="Open Poll" holder="/10" presentationState="/24" uiComponents="/12 /25"/>
              <apsm:OperationTrigger name="open" uiComponents="/13 /14" presentation="/11"/>
              <apsm>SelectOne name="Elections"/>
              <apsm>SelectOne name="Polls"/>
              <apsm:Presentation name="welcome" holder="/0"/>
              <apsm:Presentation name="Login" holder="/15" presentationState="/27" uiComponents="/17"/>
              <apsm:OperationTrigger name="Login" uiComponents="/18 /19" presentation="/16"/>
              <apsm:InputField name="Username"/>
              <apsm>PasswordField name="Password"/>
              <apsm:StateMachine name="Voting.ems.graphml" useCase="/46" states="/21 /32 /34 /66 /67 /74 /80 /22 /87"/>
              <apsm:State name="Show Ballot" stateMachine="/20" outgoing="/29"/>
              <apsm:State name="Enter Voter" stateMachine="/20" outgoing="/62" incomings="/86"/>
              <apsm:StateMachine name="OpenPoll.ems.graphml" useCase="/109" states="/24 /98 /100 /113 /119 /123 /178"/>
              <apsm:State name="Open Poll" stateMachine="/23" presentation="/11" outgoing="/31" incomings="/102"/>
              <apsm:OperationTrigger name="logout" presentation="/11"/>
              <apsm:StateMachine name="Login.ems.graphml" useCase="/138" states="/126 /128 /158 /163 /27 /153 /167"/>
              <apsm:State name="Login" stateMachine="/26" presentation="/16" outgoing="/147" incomings="/168"/>

```

Figure 66 A snapshot of the *apsm* created by running *pim-to-apsm* QVT relations

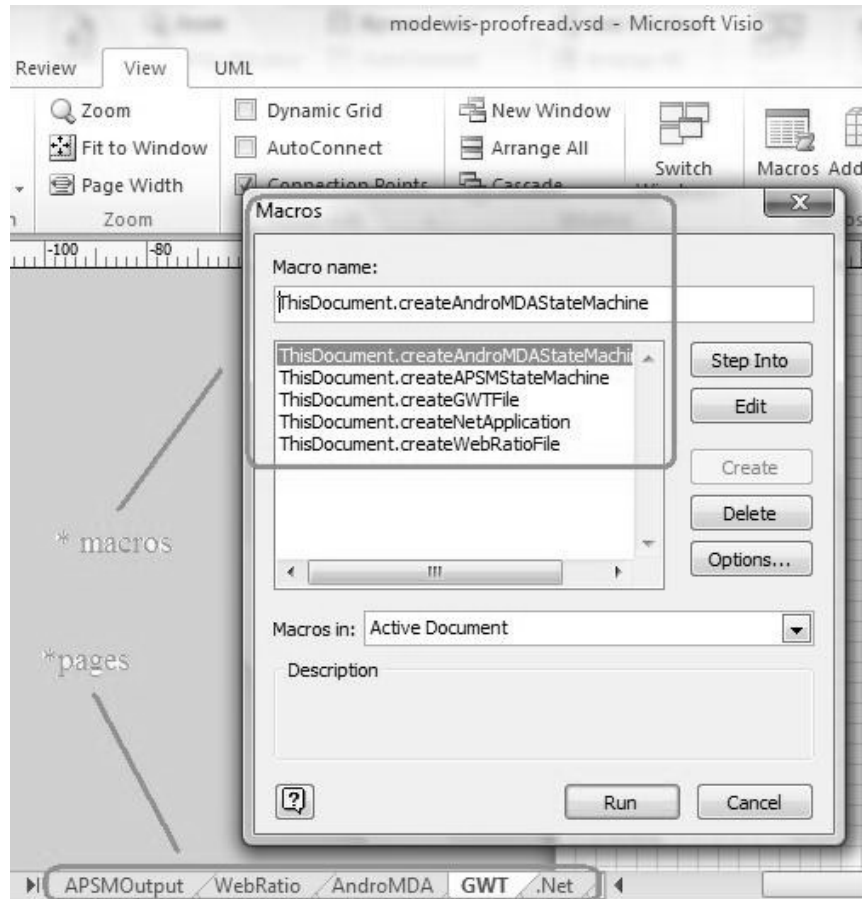


Figure 67 Pages and Macros of the ProofRead tool

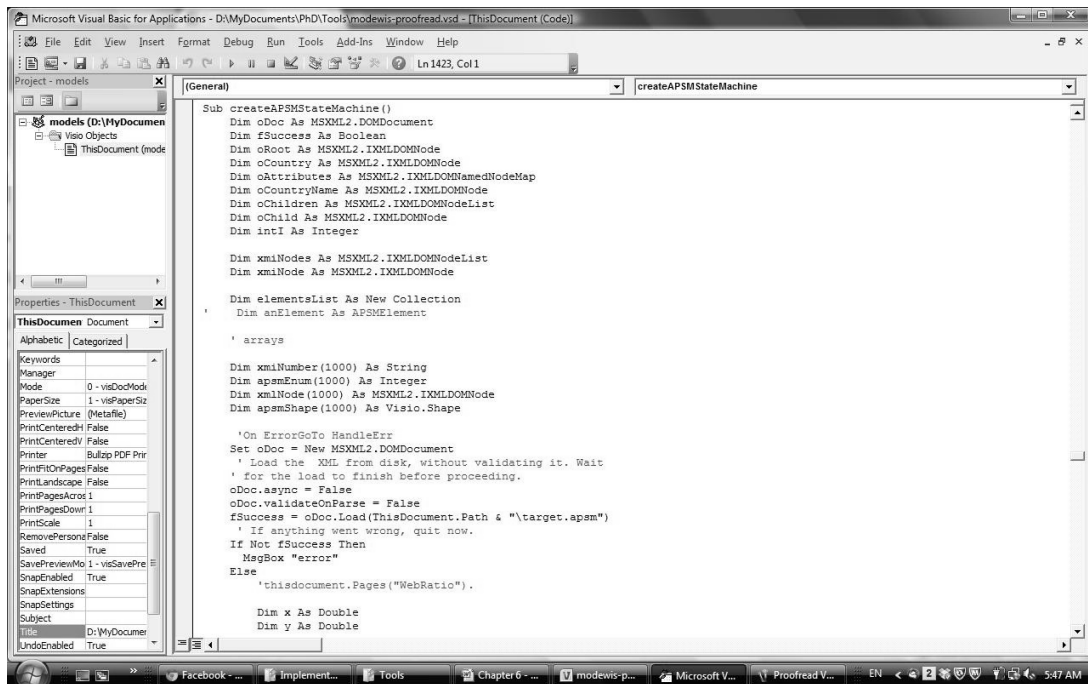


Figure 68 A snapshot of the apsm ProofRead Macro in Visio

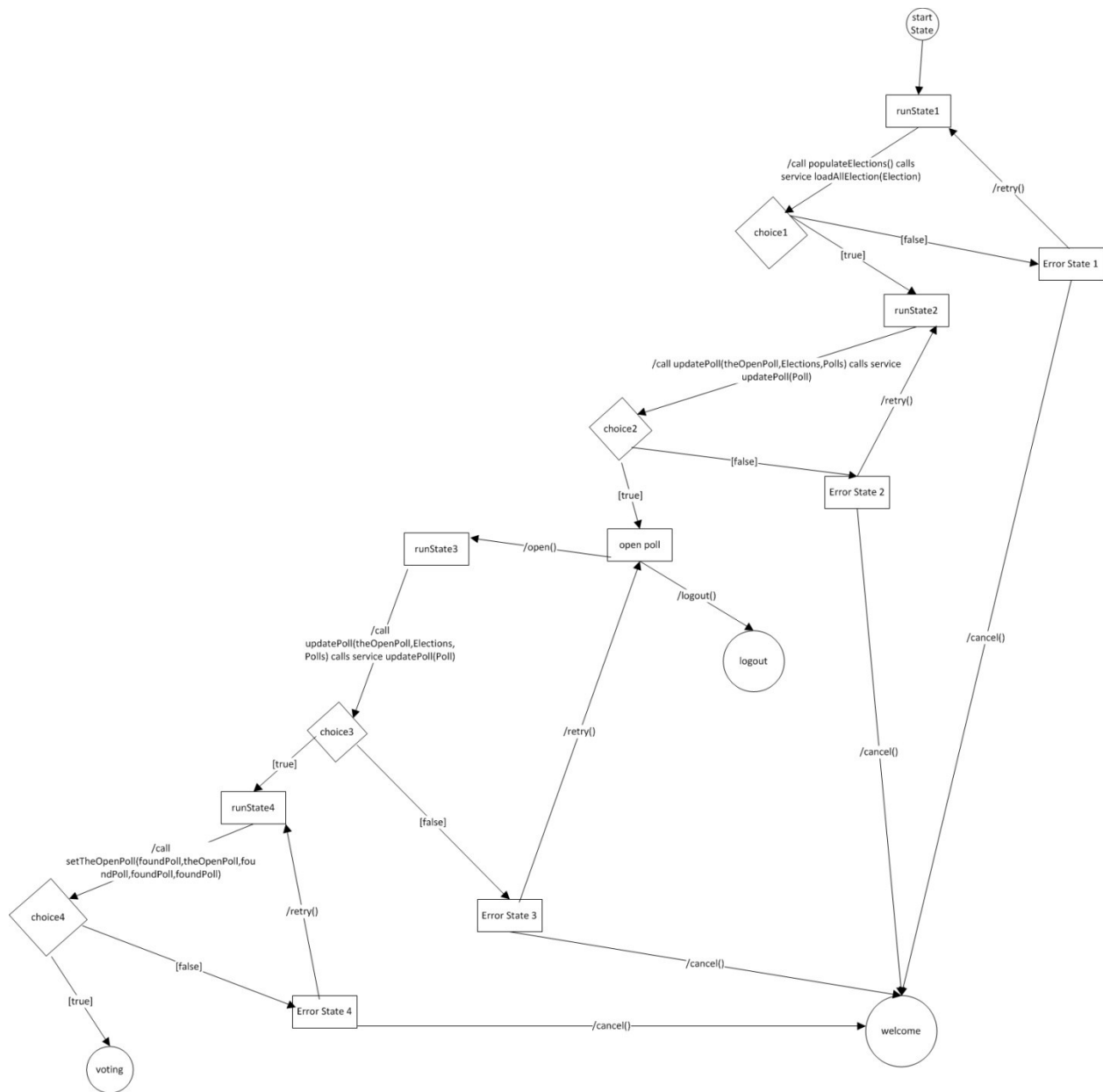


Figure 69 Proofreading results of the *apsm* open poll state machine

8.6 Mapping to GWT

Similar to the APSM and the PIM, a meta-model for GWT applications is also created and maintained under Eclipse. Figure 70 shows the GWT meta-model in Eclipse. Note that this model is not an official meta-model but just an EMF model designed to support GWT applications based on the configurations defined in Section 7.2.2. Transformations to create a *gwt* model from *apsm* are implemented using QVT relations. A snapshot of the *apsm_gwt* transformations and its run configurations is shown in Figure 71. Figure 72 shows a snapshot of

the generated *gwt* file. This file can be proofread using the GWT macro of the proofreading utility. A snapshot of this macro is shown in Figure 73. Finally, the GWT code generator may be used to generate code or a developer may choose to manually implement the generated model. Figure 74 shows a snapshot of the GWT code generator in Eclipse and Figure 75 shows a snapshot of the created GWT project.

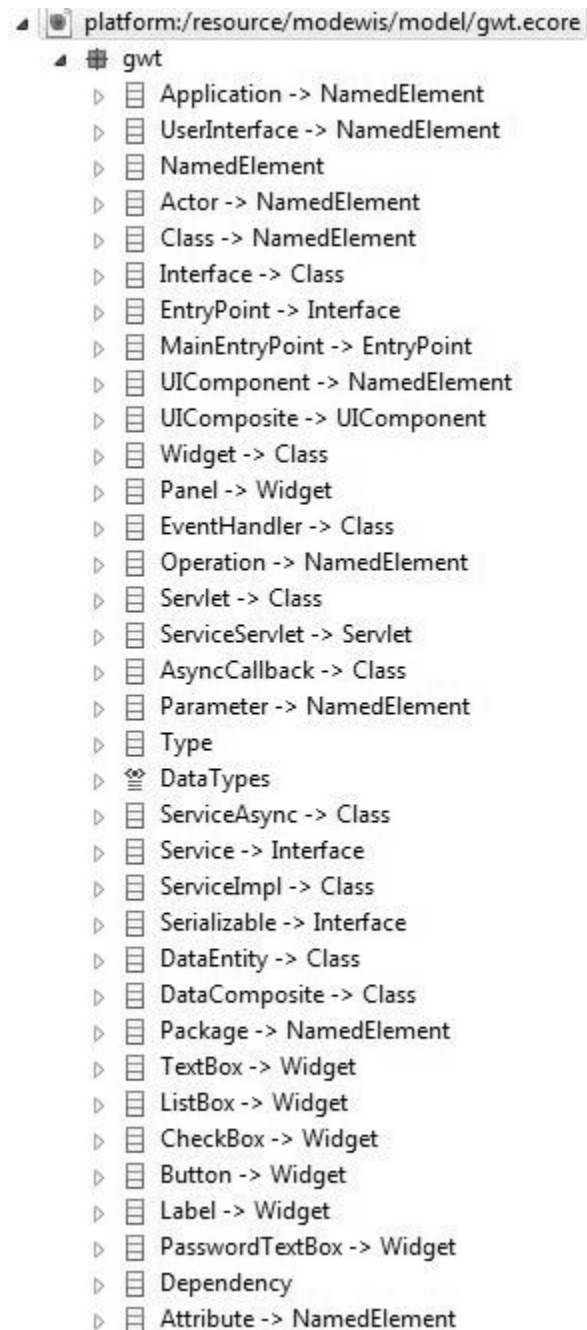


Figure 70 GWT Meta-Model in Eclipse

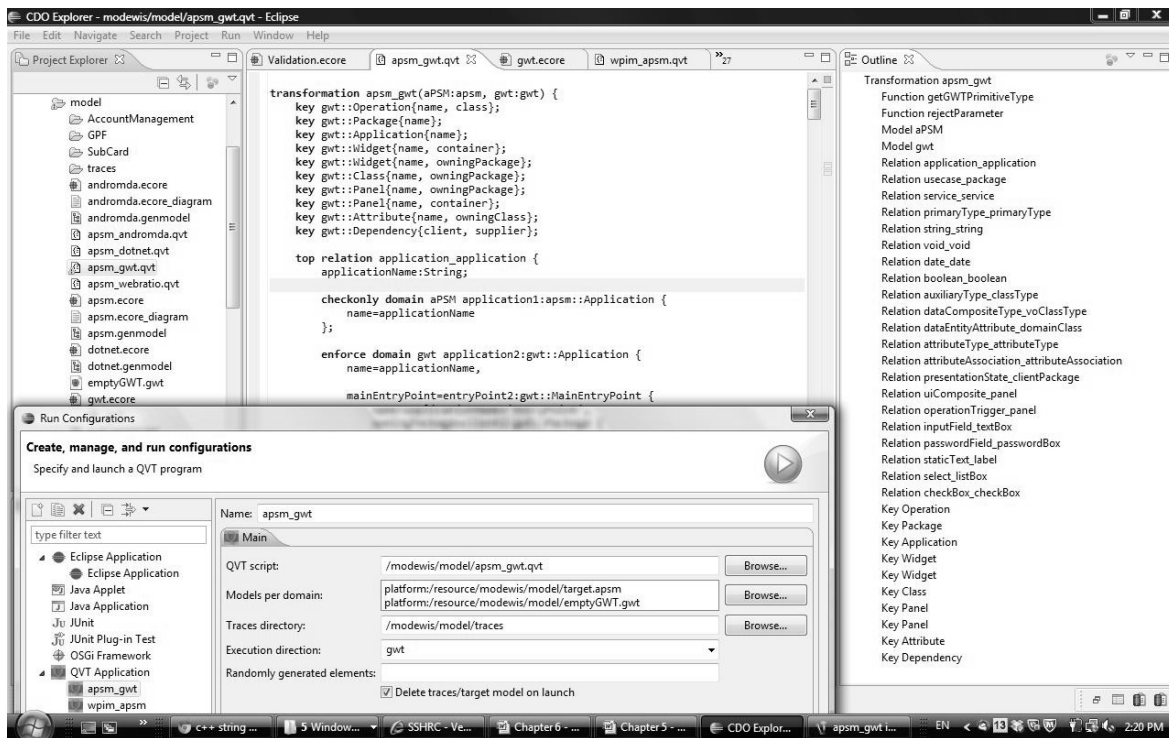


Figure 71 apsm_gwt relations under Eclipse MediniQVT

```
<gwt:AsyncCallback name="FindUserCallback" owningPackage="/11" parameter="/62"/>
<gwt:Type dataType="void"/>
<gwt:Type classType="/81"/>
<gwt:ServiceImpl name="TermServiceImpl" owningPackage="/58" clientDependencies="/90" implements="/84" operations="/79"/>
<gwt:Operation name="findTerm" class="/78" returnType="/82" parameters="/80"/>
<gwt:Parameter name="Term" type="/77"/>
<gwt:Class name="Term" owningPackage="/27"/>
<gwt:Type dataType="void" classType="/81"/>
<gwt:Parameter name="Term" type="/77"/>
<gwt:Service name="TermService" owningPackage="/1" operations="/85" asyncService="/86"/>
<gwt:Operation name="findTerm" class="/84" returnType="/82" parameters="/83"/>
<gwt:ServiceAsync name="TermService" owningPackage="/1" operations="/91"/>
<gwt:Package name="Voting.ems.graphml" owningPackage="/1" classes="/88/95"/>
<gwt:Class name="Voting.ems.graphmlController" owningPackage="/87" supplierDependencies="/90" operations="/89"/>
<gwt:Operation name="findTerm" class="/88" returnType="/82" callee="/79"/>
<gwt:Dependency client="/88" supplier="/78"/>
<gwt:Operation name="findTerm" class="/86" returnType="/96" parameters="/92/93"/>
<gwt:Parameter name="Term" type="/82"/>
<gwt:Parameter name="findTermCallback" type="/94"/>
<gwt:Type classType="/95"/>
<gwt:AsyncCallback name="FindTermCallback" owningPackage="/87" parameter="/81"/>
<gwt:Type dataType="void"/>
<gwt:Type dataType="void"/>
<gwt:Operation name="onClick" class="/68" returnType="/103" callee="/101"/>
<gwt:Type dataType="void"/>
<gwt:Button name="LoginButton" owningPackage="/11" eventHandlers="/68" container="/10"/>
<gwt:Operation name="setLoggedInUser" class="/68" returnType="/102"/>
<gwt:Type dataType="void"/>
<gwt:Type dataType="void"/>
<gwt:MainEntryPoint name="testEntryPoint" owningPackage="/1"/>
```

Figure 72 Snapshot of the automatically generated gwt file

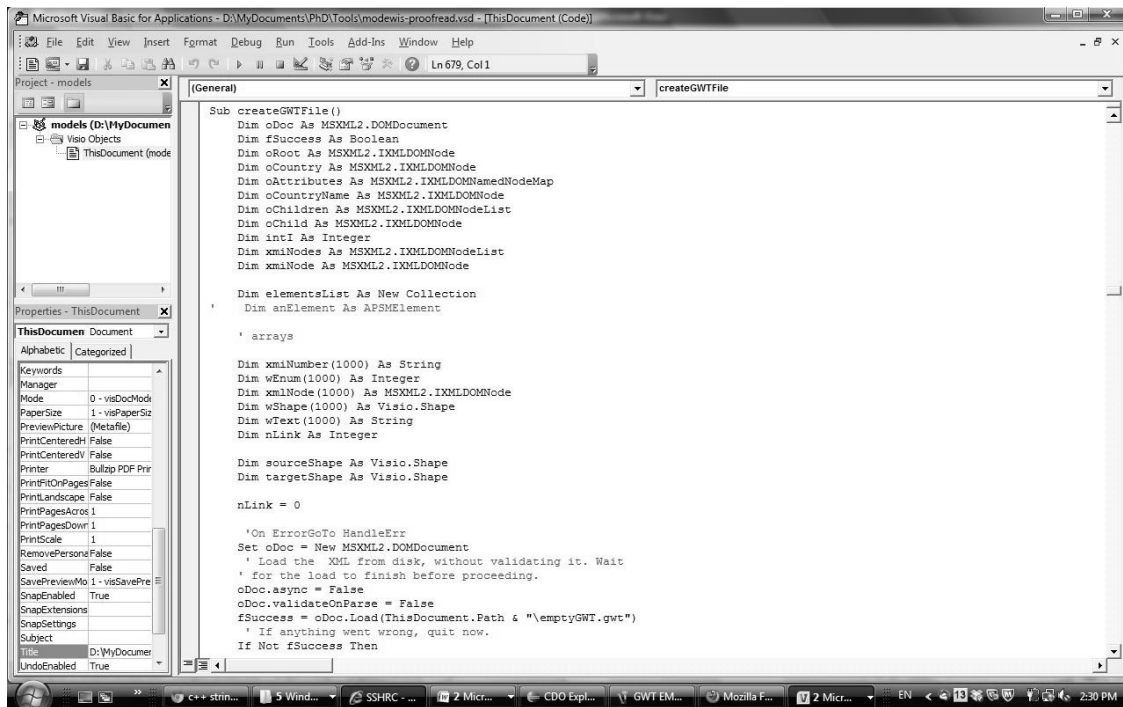


Figure 73 Snapshot of the gwt proofread macro in Visio

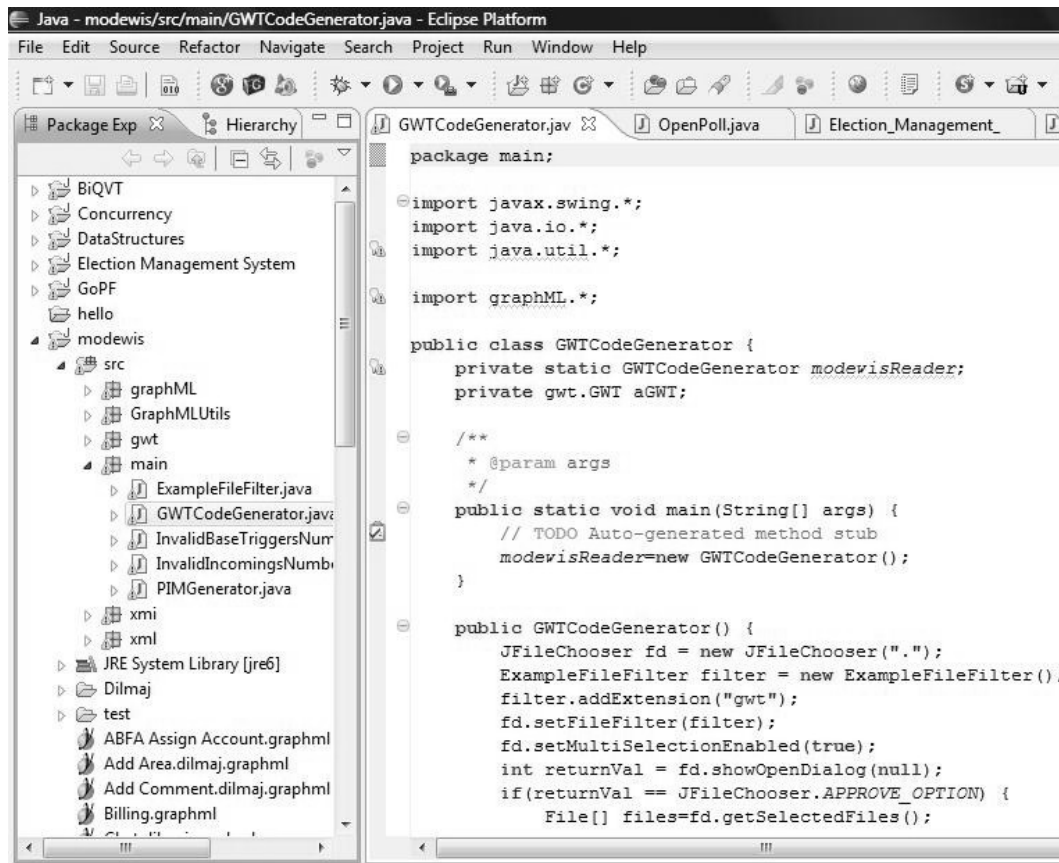


Figure 74 Snapshot of the GWT code generator project

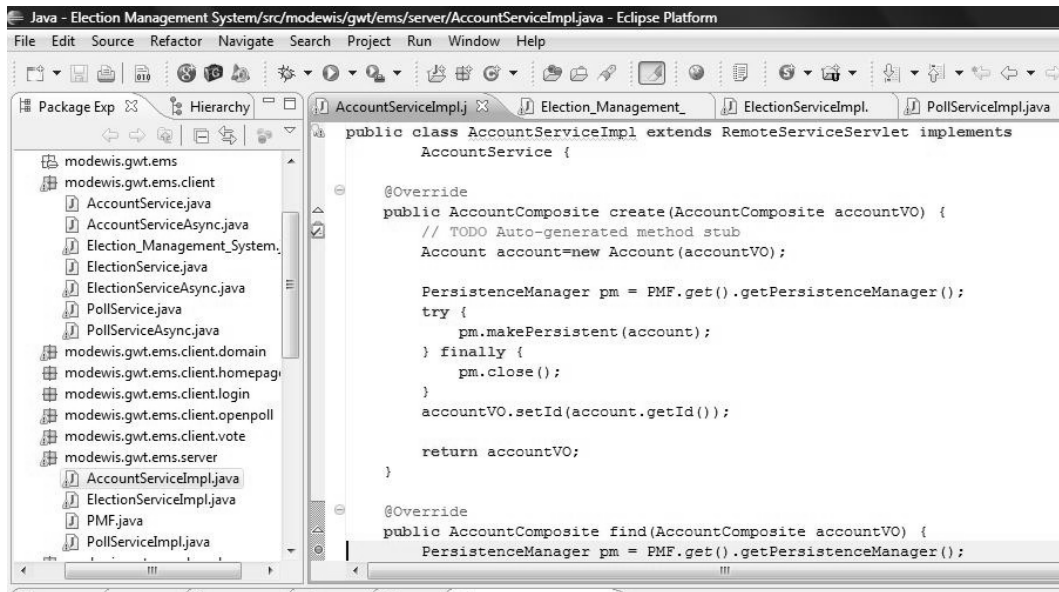


Figure 75 The GWT code of the Election Management System

Similar processes and utilities are used for other specific platforms. Figure 76 shows snapshots of the EMS while running on Google Application Engine. Figure 77 shows the same application – developed using AndroMDA – running on JBoss.

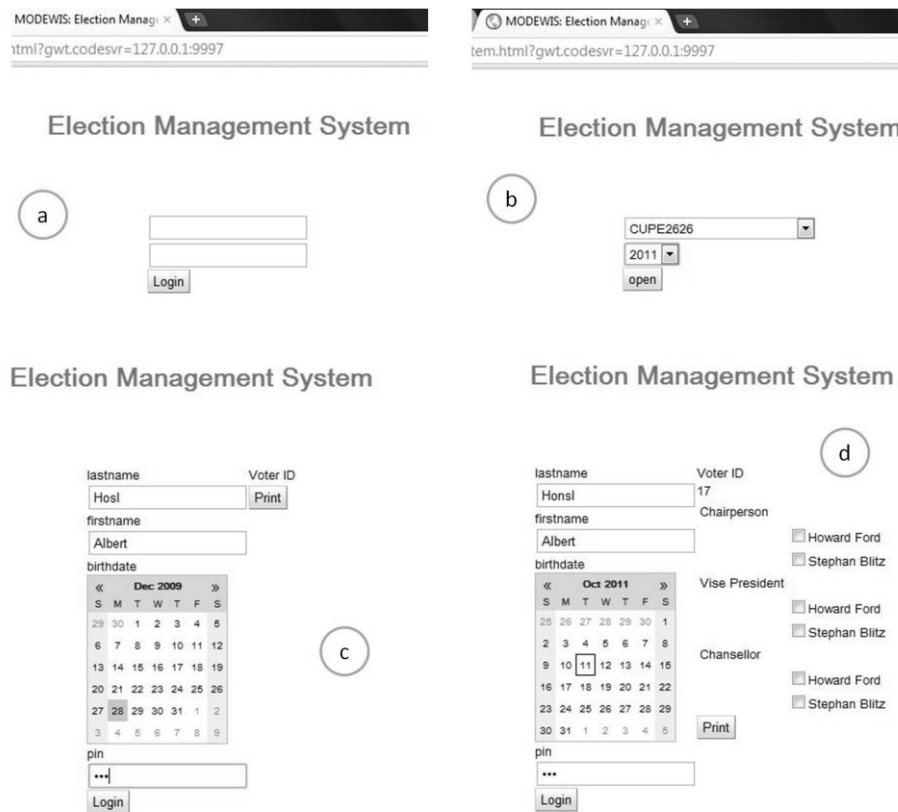


Figure 76 Snapshots of GWT EMS running on Google Application Engine

The figure displays four distinct web application interfaces for the AndroMDA EMS running on JBoss:

- Login:** A form with a title bar 'Login', a sub-header 'Login', and input fields for 'Username' and 'Password'. A 'Login' button is positioned below the password field.
- Show Elections List:** A form with a title bar 'Show Elections List', a sub-header 'Select This Election', an 'Election' dropdown menu currently showing 'CUPE 2626', and two 'Select This Election' buttons.
- Open Poll View:** A form with a title bar 'Open Poll View', a sub-header 'Open Poll', a 'Poll' dropdown menu showing 'May 2009', an 'Election' input field showing 'CUPE 2626', and two 'Open Poll' buttons.
- Vote Login:** A form with a title bar 'Vote Login', a sub-header 'Add candidate', and input fields for 'Election' (CUPE 2626), 'Poll' (May 2009), and 'Seat' (SITE Steward). It also features a 'Candidates' dropdown menu with 'Jim' and 'Mani' visible, and two 'Add candidate' buttons.

Figure 77 Snapshot of the AndroMDA EMS running on JBoss

Figure 78 is a snapshot of MagicDraw that presents the automatically generated code for the AndroMDA-specific platform. The code in Figure 78 belongs to the *createUsers* operation of the controller class generated for the use case *Register User*:

- 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 7.1.1.
- 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 (Rule 7, Section Chapter 7); the returned value is decided based on the return value from the pertinent service operations in Lines 13-14.

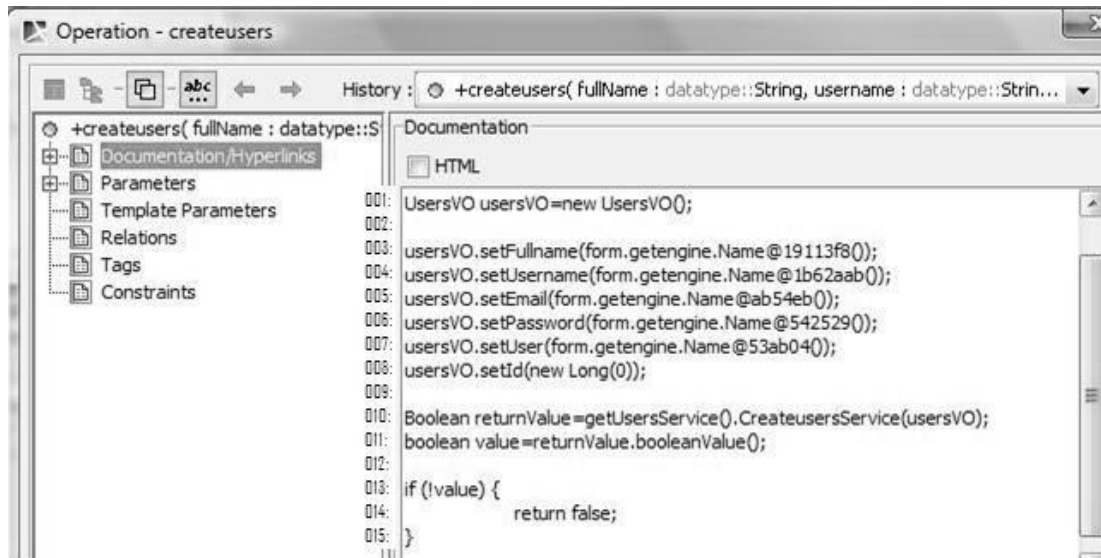


Figure 78 A snapshot of MagicDraw showing automatically generated code.

8.7 Summary

In this chapter, a visual overview of the implemented approach was presented. We reviewed different steps required to generate an executable application for multiple platforms from a visual *pim*. More details are found in Appendix D and in [MDWSQVT2011].

Chapter 9. EVALUATION

In this chapter, we evaluate the ASPM from different perspectives. In Section 9.1, the ASPM is examined against the criteria set in Section 3.1. More details on the evaluation addressing platform-independence, separation of concerns and requirements are presented in Sections 9.2, 9.3, and 9.4. The ASPM, being an abstract model, must show the capability of being mapped to multiple platforms. Four different platforms were specified and formal transformations were provided. The transformations were implemented as QVT relations. A set of utilities were used to set up the PIM and to generate platform-specific code. Furthermore, a proofreading tool for the QVT relations in this thesis was implemented. This implementation was described in Chapter 8 and is examined from a software engineering point-of-view in Section 9.5. Finally, in Section 9.6, we present the results of the application of our approach to case studies.

9.1 *Abstract Web Criteria*

The ASPM successfully satisfies the requirements discussed in Section 3.1 as follows:

- **Support for data access and event handling.** We provide operational support from presentation layer to data-access layer facilitated by controllers that dispatch triggers throughout different data services. The transmission of data between layers is supported by data composites while the data itself is modeled using data entities.
- **Platform-Independence.** Our model remains at the ASPM level and is yet adaptable to other SPSMs; this ensures platform-independence. Our mappings to sample platforms show that the meta-model can map to multiple platforms. We were able to support platforms with different features using our transformations. For example, consider the controller mechanism. While our AndroMDA and .Net-based platforms support controllers directly, WebRatio and GWT do not specifically support the controller pattern. But we have been able to specify transformations that map the ASPM controller pattern to specific controlling features of both GWT and WebRatio. This is done by mapping the controller feature to distributed operation units in WebRatio, and controlling operations accessed through entry points in GWT.

- **Support for MDD necessities.** We use concrete associations between different elements to ensure the required traceability by MDD transformations independent of naming. For example, the concrete association instance between a *UIComponent* and a *DataComposite* provides the same type of support as suggested by Cichetti and Di Ruscio [Cich2007] in terms of an association from *DataCompositionWLink* and *CompositionWElement*. We also provided an input language in Section 5.1, which is used to define the PIM and hence potentially simplifies the usage of the APSM. Transformations from this input language to the APSM were also presented in Section 6.1.
- **UML support.** Our model is an extension of a UML-based model [Botter2007] according to the UML extension mechanisms. This is important to promote the usage of the APSM within the UML community and to be able to interact with UML-based tools and methods.
- **Support for Security.** Our model pays specific attention to authentication, session mechanisms, and secure access. As described in Section 4.7, the APSM also presents a built-in authentication package, which may be used as it is or altered when mapping to specific platforms.
- **Support for Web 2.0.** We consider elements that help develop applications with Web 2.0 features. Two main features of our model that help support Web 2.0 are: 1) a presentation state may be composed of several presentation units, which allows independent update of different units within the same page; and 2) content-oriented UI elements accept interactions such as comments, feedback and ranks, as required by Web 2.0. Furthermore, content-oriented UI elements are attached with search interfaces that imply a fully searchable content.
- **Support for expressing requirements.** Our approach closely relates requirements to lower-level models and hence increases the chance of developing applications in accordance with requirements.
 - Use case behaviours are modeled using state machines. Several features are provided for the mappings of different use case concepts such as inclusion/extension, use case steps and scenarios directly to state machines. The

resultant *apsm* state machines inherit the same type of relationships that exist among use cases.

- We consider UI prototypes as part of requirements. These UI prototypes are modeled along with the behaviour of use cases to form an integrated input to the approach.
- We also include information elements within the description of UI prototypes as well as query marks used to elaborate the relationships between a UI element and associated information elements. This information is used for the automated generation of data-related operations.
- **Separation of Concerns.** Separation of concerns is supported by the APSM. The APSM allows modeling application in different layers of presentation, control and data access.
- **Web-Specific Features.** Different web specific features are also supported by the APSM:
 - Structures that are specific to the Web such as Web forms and hyperlinks are embedded in APSM.
 - UI elements are supported at an abstract level; that is the abstract web model does not detail many aspects such as color, size and position of web elements.
 - UI components such as buttons, inputs and tables are supported.
 - UI components are integrated with the web application so one can model the interactions of presentation and controller at an abstract level.
- **Availability of Details.** Appendix A describes the details of all elements in the APSM.

9.2 Mapping to Multiple Platforms

The APSM is designed to be adaptable to more than one specific platform. In this thesis, four specific platforms were used as SPSMs. Two sets of transformations are needed from PIM to SPSM: a first one to map PIM to APSM and the second one to map APSM to SPSM. The PIM-to-APSM part of the transformations is reusable for all applications as both sides of the mapping are independent of the specific platforms. In order to create an application from the PIM, only the QVT relations required for generating a *spsm* from the APSM need to be supplied. Since the APSM is semantically closer to PSMs, this latter part is expected to be easier to develop in comparison to the conventional PIM-to-PSM transformations.

Since an objective of model-driven development is to automate parts of the software engineering process, it was important to show the possibility to derive the full specification of an application from a minimum set of descriptions. Having the APSM automatically generated using the PIM-to-APSM transformations, the platform-specific part of the transformations, i.e. transformations known as APSM-to-SPSMs, is kept at the minimum. In order to evaluate the meta-model for specifying an abstract web application in a model-driven context, formal transformations to all four platforms were provided using QVT Relations:

- 1) APSM-to-AndroMDA: This mapping creates a fully detailed AndroMDA model, which can be transformed to code using AndroMDA.
- 2) APSM-to-WebML: The target in this mapping is a specific configuration of the WebML-based tool, WebRatio.
- 3) APSM-to-GWT: This maps the APSM to a platform, based on Google Web Toolkit. This validates the Web 2.0 modeling capabilities of the APSM.
- 4) APSM-to-Net: The target of this set is to map the APSM to a platform based on Microsoft .Net.

Different types of transformations were required for each target. The mappings to AndroMDA belong to the category of refining mappings according to Mellor et al. [Mellor2004] because both the APSM and AndroMDA are defined based on UML. Mappings to WebRatio and GWT were rather more complicated to design because the target has a different structure than the source. These types of transformations may be called *Migrating* transformations – according to Mellor et al. [Mellor2004] – that copy the source model to a completely different format at the target – see Section 2.6.3 for more details on refining and migrating transformations. The set of transformations discussed in this thesis contributes to the validation in different ways.

1. Platform-Independence: By mapping to more than one specific platform, we show that the APSM is mappable to more than one specific platform. The existing web models are generally validated by providing mappings to one specific platform.
2. Correctness: The fact, that QVT relations compile and run successfully, supports that the meta-models in this thesis, including the APSM, are syntactically valid and contain no errors.

3. **Saving by Reusability:** By creating two types of transformations, the APSM changes the conventional way of creating transformations. In conventional MDD, the transformation from PIM to PSM is essentially platform-specific and may not be used with the platforms other than the one for which it was designed. Our PIM-to-APSM transformation is reusable and only the APSM-to-SPSM transformations must be created for a new web platform. This results in a better value because models are made more valuable products. Models form the value of model-driven development (MDD) methods. Transformations add to the value of MDD by processing raw models and by creating more detailed ones that could be used for the development of actual applications. We maximize the level of value re-use while minimizing the length of transformations that create value or add to it.

9.3 Separation of Concerns

The number of layers in the APSM is not limited. For example, the presentation layer could be divided into 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 – using our approach – 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.

Figure 79 shows an example of nested layers designed using the APSM. As this figure shows, there are two levels of design, each pertinent to a separate use case; each layer contains its own presentation and data elements. *UseCase1* is a summary-level use case, while *useCase2* is a user-level use case – as defined by Cockburn [Coc2001], see Section 2.3. As Figure 79 shows, the presentation layer *TutorialComposite* is composed of lower-level presentations *FeatureComposite* that are connected to the logic prescribed by step use cases. This is an example of the online tutorial of an application that contains links to actual use cases of the application. Use case steps may be activated either individually or from within the tutorial. This example shows how the APSM supports designing an online tutorial, where the presentation, denoted by the UIComposite *TutorialCompsite* is controlled by a controller *TutorialController*

that is different than the ones belonging to tutorial steps, i.e. *FeatureComposite* and *ApplicationController*. Thus, the designer has the option of breaking the top-layer presentation *TutorialComposite* to other presentations namely *FeatureComposite* and give each layer its own specific controller.

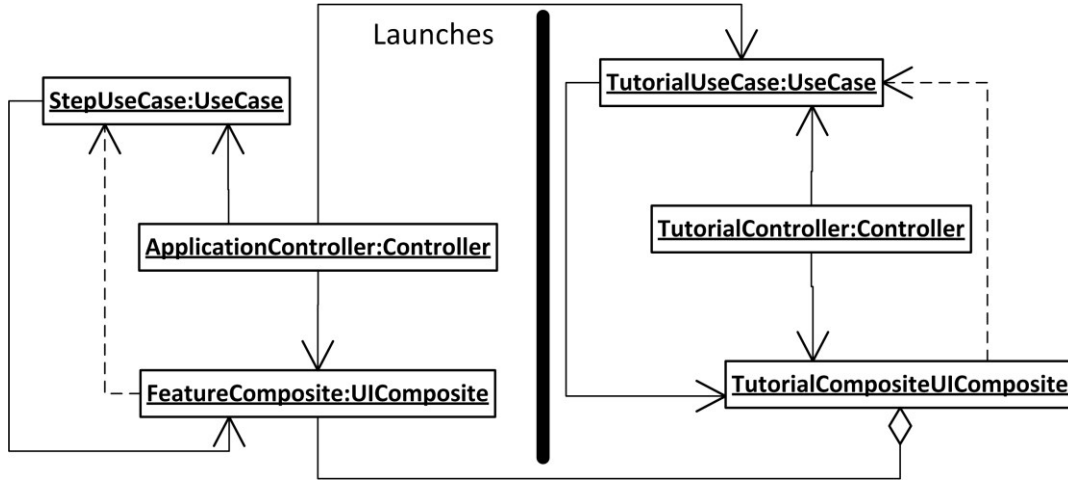


Figure 79 Flexibility of Architectural Layers in an Online Tutorial

In Figure 79, *StepUseCase* and *TutorialUseCase* are instances of *UseCase*, *ApplicationController* and *TutorialController* are instances of *Controller* while *FeatureComposite* and *TutorialComposite* are instances of UI composite. Figure 80 presents an object model for the same example. In this Figure, the use case *main scenario* has two steps, *step 2* and *step 3*. Note that since the use case *main scenario* is a summary-level use case, its steps can be associated with user-level use cases and APSM supports this. While step 2 is a simple use case step, step 3 has its own user interface *presentation2*, which displays the contents of *dataComposite2*. The *main scenario* itself deals with *presentation1* that displays *dataComposite1*. Once step 3 of the use case is selected, the content of *dataComposite1* is used to select and show the contents of *dataComposite2*. Note that *presentation2* is invisible at first.

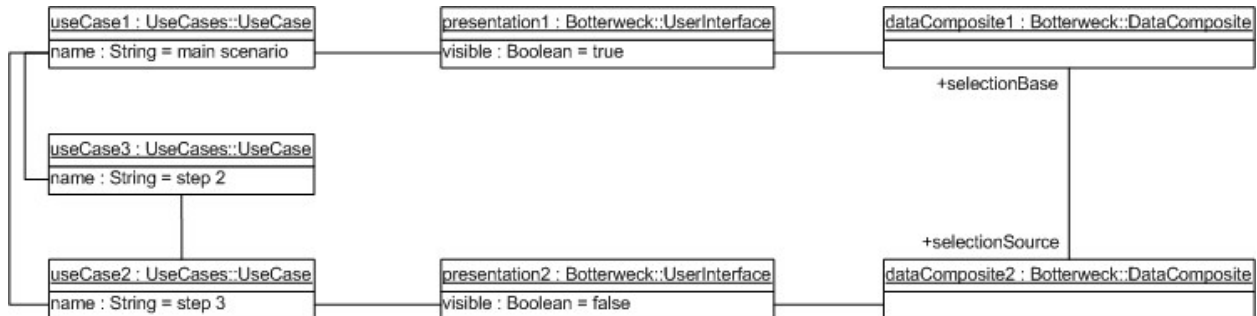


Figure 80 Composite data used on composite UI.

Finally, our .Net-specific platform supports defining applications in seven layers, hence validating the capability of mapping to specific platforms with more than three layers. These layers are:

1. Presentation layer consisting web pages and use controls.
2. Unit-specific logic performed at the code-behind of each page or user control.
3. Centralized logic of each use case contained in use case controllers.
4. Explicit data requests performed at data services.
5. Parameterized data requests performed at data-specific accessors.
6. Generic data requests performed at the application-specific generic data access.
7. Concrete SQL statements running in stored procedures.

9.4 Handling Requirements

The APSM is aimed at incorporating models that relate high-level requirements to lower-level design models. Supporting tools and processes do not directly address requirements elicitation. These tools, however, create automated mappings between requirements models, i.e. state machines and UI prototypes, and design models. In this section, we briefly review the relationship of the APSM and its supporting tools with requirements models.

Requirements are generally known as being either functional or non-functional. Functional requirements are satisfied by the behaviours of specific components of the software [Bray2002]. Functional requirements can be expressed by use cases and the models detailing the behaviour of use cases. Therefore, functional requirements are vastly covered by the APSM in the sense that it is based on use cases. Again, it is worth mentioning that although APSM addresses the requirements described using use case behaviour models, we cannot make any claim regarding the correctness or completeness of those requirements.

Another group of requirements are known as non-functional requirements. Lethbridge and Laganière [Leth2001] divide non-functional requirements into quality, platform and process requirements, including the requirements regarding the speed, capacity, reliability and usability of the solution as well as those addressing the constraints regarding design, commercialization and other preferences. Some non-functional requirements could be operationalized within use

case and behaviour models. For example, the fact that we use a choice vertex in a state machine (or an alternative step in a use case) to check whether a User has reached a maximum allowed number of unsuccessful login attempts could be considered as the expression of an operationalized non-functional requirement related to security.

The use of the APSM could partially help realize some of the non-functional requirements. The APSM is at least partially concerned with usability as we incorporate the desired user interface as a requirement and we use it as an input to our method. For other types of requirements, we highly rely on the use of MVC, MDA and the chosen implementation platforms. The usage of MVC can partially address some non-functional requirements in a general context such as reliability and flexibility through the separation of concerns. Some other non-functional requirements are tightly associated with the chosen implementation platforms. For example, immediate update of web components is possible in our GWT platform while it cannot be achieved without refreshing the page in our .Net platform because we have not included Javascript/Ajax in the latter platform.

9.5 Methodological Evaluation

In order to evaluate the APSM in the context of a software engineering process, a specific approach is introduced in this section. We also compare this process with the conventional software engineering approaches from a theoretical and practical perspective.

Figure 81 depicts a general overview of the 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 the *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.

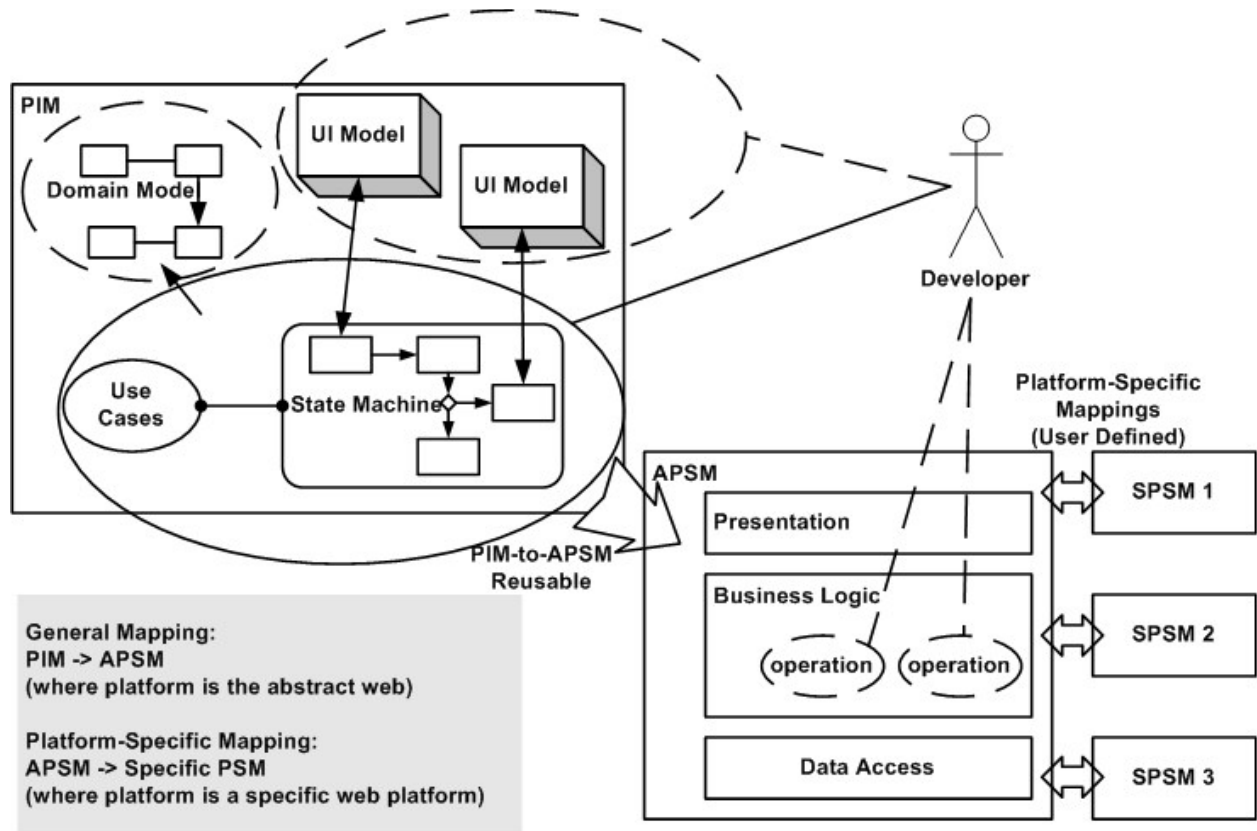


Figure 81 MODEWIS: A Semi-Automated Approach

We now compare MODEWIS with conventional software engineering approaches. We start by recognizing the type of activities done in MODEWIS and comparing to the ones of theoretical software engineering. According to Lethbridge and Laganière [Leth2001], a typical software engineering process consists of the following activities:

- Requirements/Specification, which consists in gathering requirements, understanding the domain, differentiating *whats* from *hows* and enlisting things the User would expect the software system to do.
- Design, where design choices such as selected technologies and comprising subsystems are made in order to ensure the satisfaction of requirements.
- Modeling, this is about visually specifying the designed system and requirements using modeling languages. Use case modeling, domain modeling and structural modeling are examples of modeling activity.

- Programming, this is the act of writing the actual code that implements the application functionality.
- Quality Assurance, this activity includes the validation of the implemented software application to check that it satisfies its requirements.
- Deployment, this process consists in releasing the application to the customer, installing required components and giving required instructions for using it.
- Management, which relates the managerial aspects of the software development including cost estimation, task planning and resource assignation.

The following is a list of activities in MODEWIS and how they relate to the conventional activities of software engineering:

- 1 PreConditions: Requirements are elicited but not necessarily modeled. Use cases are realized and are optionally documented as textual descriptions. Data and usability requirements are considered so the developer can decide which data is used in every use case step as well as what is presented in every presentation step.
- 2 PIMing. A *pim* serves as the input to the process and is composed of use case state machines and UI prototypes. This step is mainly manual. At some points, the developer will need to assign use cases as next steps assigned to states of the state machine or transitions assigned to operation triggers. Such activities are helped by automatically created lists of cross-cutting use cases, triggers and states. PIMing is currently performed in two threads:

2.1 First, the *pim* is modeled using *yEd* and the collection of the symbols created for the purpose of PIMing.

2.2 Second, the modeled *pim* is mapped to EMF using the MODEWIS_PIM_Generator utility.

The two steps can be merged in future to come up with a modeling tool that generates EMF.

- 3 APSMing. The *pim* is mapped to an *apsm* using the PIM_APSM set of QVT relations. The output can be investigated using the APSM section of *MODEWIS_ProofRead*.

- 4 SPSMing. The resultant *apsm* is mapped to one of the specific platform. This can also be tested using one of the specific sections of *MODEWIS_ProofRead*.
- 5 Coding. Code is generated using code generators that accept EMF as input.
- 6 PostConditions. The generated code might need to be manually edited to account for specific aspects of the target platform.

Table 15 compares software engineering activities with the steps of MODEWIS:

Table 15. MODEWIS Steps vs. Software Engineering Activities						
	Pre	PIMing	APSMing	SPSMing	Code Generation	Post
Requirements/ Specification						
Design				Specific Platform is defined here		
Modeling		UI Prototyping and Use Case modeling using state machines	Auto-matically Done	Automatically done		
Programming					Auto-matically done	Manual coding required
Quality Assurance			Proof-reading Transformation Results	Proofreading Transformation Results		
Deployment						*
Management					N/A	

As Table 15 shows:

- Requirements/Specification is out of the scope and is considered a prerequisite of our approach. More specifically, functional requirements, usability expectations and domain data must be determined before the process starts.
- Design affects our approach especially in terms of selecting and setting up the specific platform(s). The specific platform may or may not be among the four examples discussed in this thesis. If the specific platform is not available, the specific platform and its related APSM-to-SPSM transformations need to be defined based on the user's selection of database, programming, deployment and other related technologies. From a business viewpoint this might be a drawback because it adds to the cost and time of development. A solution could be adding as many specific platforms as possible based on the most common technological configurations. This will be a part of the future work.
- Modeling is where the largest part of the manual work resides. The UI and use cases that serve as the input to our approach are modeled here. However, the input is provided as a minimal subset of the meta-model, i.e. the PIM. The rest of the modeling is performed in the context of automated transformations that create the *apsm* and relevant *spsms*.
- Programming is supported by the executable application being automatically generated using code generators. This involves all data processing operations and many other logical operations performed for checking the validity of the input and perform web-specific tasks such as sending emails and comparing input values. **Manual coding is required to give the application a more user-friendly look-and-feel as well as to improve performance.** Nevertheless, it should be noted that code generation is only performed as a proof-of-concept and we cannot make any claim regarding the suitability or completeness of the generated code.
- Quality Assurance (QA) is performed against the models and transformations. In a typical software engineering process, the focus of quality assurance is mostly on the quality of code. But in our approach, we focus on models. Modules have been created to generate a schematic diagram of the application based on the platform-specific models, i.e. *apsm* and *spsms*. These diagrams are a visual assistance to reviews aiming at showing the correctness of the models resulting from the transformations.

Such QA can be helpful when developing new transformations or changing the existing ones to ensure the transformation is designed correctly. It can also help developers verify the resultant abstract or specific application before generating code and to change the inputs as required.

9.5.1 A Sample Use Case

We present an example based on *The NextGen POS System* [Larman2005] in order to compare system development using MODEWIS to one of the conventional software engineering approaches. The *NextGen POS System* aims to assist cashiers in a department store. The main scenario of the use case *Process Sale* is presented in Listing 17.

Listing 17 Main Scenario of the Process Sale Use Case [Larman2005]

```
1- Customer arrives at a POS checkout with items to purchase.
2- Cashier starts a new sale.
3- Cashier enters item identifier.
4- System records sale line item and presents item description, price, and running total. Price
   calculated from a set of price rules.
5- Cashier repeats steps 3-4 until indicates done.
6- System represents total with taxes calculated.
7- Cashier tells customer the total and asks for payment.
8- Customer pays and System handles payment.
9- The Cashier tells the customer the total.
10- System logs completed sale and sends sale and payment information to the external Accounting
    system.
11- System presents receipt.
12- Customer leaves with receipt and goods.
```

Figure 82 shows the main scenario of the use case, *Process Sale* [Larman2005] and its corresponding UI prototype depicted as a part of MODEWIS:

Operation Contracts

As a part of the process, Larman's approach creates operation contracts. An operation contract specifies an operation preconditions and post-conditions in terms of the objects and variables used at the input and output. They provide a basis for creating sequence diagrams and change use cases from plain text to valuable source of information for systematic design. Operation contracts are keys to Larman's approach. Table 16 shows operation contracts that Larman's approach suggests for the use case, *Process Sale* compared to the operations automatically created by MODEWIS. These MODEWIS operations are created for the controller of the use case *Process Sale*.

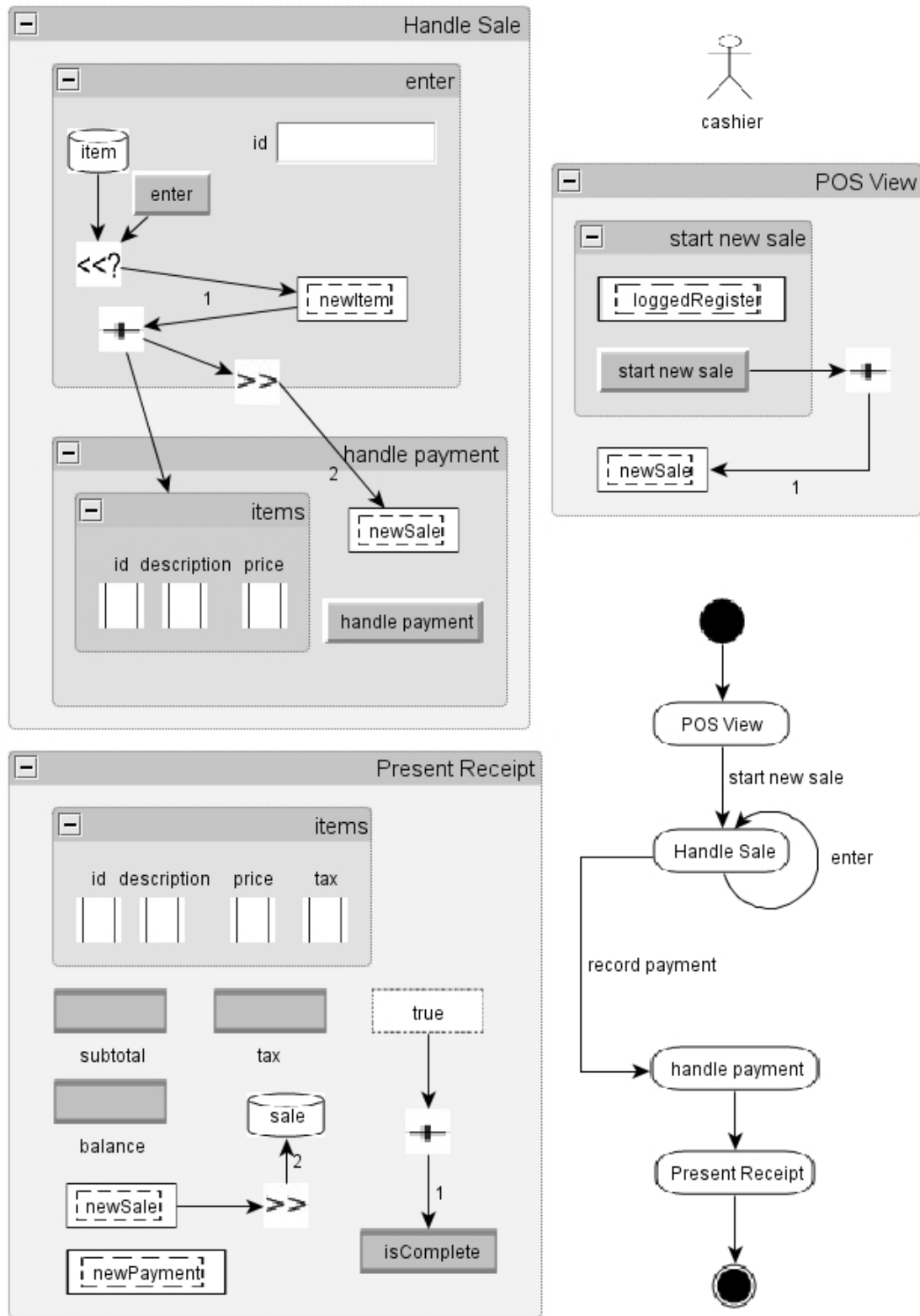


Figure 82 The PIM of the Use Case, Process Sale [Larman2005]

Table 16. Larman's Operation Contracts vs. Controller Operations of the use case, Process Sale		
Larman's Approach		MODEWIS
Operation Contract	Postconditions	Controller Operations
1- makeNewSale	11- A Sale instance s was created 12- s was associated with a Register 13- Attributes of s were initialized	setNewSale(newSale, loggedRegister)
2- enterItem	21- A SalesLineItem instance sli was created 22- sli was associated with the current sale 23- sli.quantity became quantity 24- sli was associated with a ProductDescription based on itemID match	populateItemsTable(id, newItem) setNewItem(newItem) addToItemsTable(newItem) updateNewSale(newItem, newSale)
3- endSale	31- Sale.isComplete became true	setIsComplete(newSale, "true") updateSale(newSale, isComplete, newPayment, items, subtotal, tax, balance)
4- makePayment	41- A Payment instance p was created 42- p.amountTendered became amount 43- p was associated with the current Sale 44- The current Sale was associated with the Store	handlePayment(newSale, items)

As Table 16 suggests, MODEWIS generates all the required operations. The generated operations might be less efficient – as several operations need to be called for a task - or accurate than the ones of Larman's but they are nevertheless sufficient. One can observe that MODEWIS generates more operations and some of these operations such as *setIsComplete* are fine-grained and may be seen as equivalent to one atomic action. Following is a description:

- *makeNewSale*: *loggedRegister* is considered an attribute of the data variable *newSale*. Hence, *newSale* is associated with *loggedRegister*. This satisfies postconditions 11, 12 and partly 13. The rest of the attributes of *newSale* are set in the context of other operations and use cases.
- *enterItem*: Operations *populateItemsTable* and *setNewItem* handle postconditions 21 and 22. The operation *populateItemsTable* also satisfies the postcondition 24 because it loads

data based on the input *id* from the data table *item*. Also having table columns *description* and *price* associates these attributes with *id*.

- *endSale*: This is handled by two operations: *setIsComplete* and *UpdateSale*. Note that the operation *updateSale* accepts the values of all components although the values of *subtotal*, *tax* and *balance* exist in the variable *newPayment* as well. Currently we have no mechanism to detect and remove this type of dependency.
- *makePayment*: The operation, *handlePayment* accepts the parameters *newSale* and *items*. Since handling payment is considered a separate use case, the output that is *newPayment* is inserted in the final presentation unit *Present Receipt*. The operation *updateSale*, which runs at the beginning of loading this state, confirms that *newPayment* should also be assigned as an attribute association with *newSale*.

9.6 Case Studies

In this section, we evaluate our work based on the implemented case studies and the results of our experiments. A main case study is a Web 2.0 application, Dilmaj [Dilmaj2010]. Dilmaj is a web-based, collaborative, multilingual project about languages and language development supported by a group of developers. Many domestic languages are challenged in modern times. Dilmaj offers a central place where users can contribute to the development of a chosen language, and its relation to other languages. Users may join, suggest terms, and comment on others' suggestions or rank entries. Models, descriptions and details of Dilmaj including the *pim* and the *apsm* can found in Appendix B.

The Dilmaj development team had found the preliminary ideas very hard to formally define and extend. An estimation of the required time to develop a working version was about three months. Instead, MODEWIS was chosen as an approach and a complete working application was developed using MODEWIS in one week. The resulting application is completely functional and the development team uses it as a venue for developing the initial collection of data to be exported to the final version. This code is available online [Sokhan2010]. Optimizations to data queries as well as the appearance of the UI are considered changes required to be done to the automatically generated application.

Besides Dilmaj, other major case studies where our approach was applied are:

- The Account Management System (AMS) [AMS2010], a project foreseen by a waste/water management company to replace an existing non-web system. The AMS has more than 40 use cases. Models related to this application can be found in Appendix C. Another evaluation was conducted with the AMS. In this evaluation, the developers were gathered in a workshop. They were provided with a brief introduction of the approach as well as samples outputs such as the one shown in Figure 78. Then they were asked to evaluate how much extra work would be required to complete a system automatically generated using MODEWIS. The result was up to 20% in terms of extra lines-of-code would have to be added after generation.
- The Election Management System (EMS) [Leth2001]; a scenario of this application is modeled using two different platforms. The details can be found in Appendix D.
- A Team Management System (TMS) [Somé2008]. The TMS was used as a course project. We compared an automatically generated TMS application with students' projects delivered as part of their course projects. The comparison shows that in most cases the generated application is equal to what students have manually developed in terms of functionality.

These case studies have been selected based on criteria such as the complexity of CRUD operations, the number of use cases, the complexity of the use case model in terms of number of inclusion/extensions and the nature of the domain.

In summary, case studies developed using MODEWIS support the functional requirements, the UI and navigation requirements as well as classes required to model services, data and the application logic [Fatol2010-5] as presented in Appendix B, Appendix C and Appendix D. Extra coding work is required to improve the efficiency, enhance the UI and provide more complicated logical behaviours.

An interesting observation is made with regard to the state machines and UI prototypes of use cases. Results show that input models for Web 2.0 applications, e.g. Appendix B, are generally composed of more sophisticated UI models while the state machines describing use cases are often simple and small. On the contrary, input models for conventional web

applications, e.g. Appendix C, contain relatively more complicated state machines and simpler UI models for presentations. The development of all case studies, using MODEWIS, was completed in up to one month depending on the complexity of the application as well as the difficulty of designing mappings from the *apsm* to the target platforms. This is in all cases shorter than the actual development time. For example, the development of the original AMS had taken about one year and a team of developers.

We estimate how much saving resulted from transformations at the modeling activity by comparing the size of the manually created *pim* file with the size of the *apsm* file automatically created from it. For an individual use case, the average saving is 68.5%, that is, on average 68.5% of the modeling work is covered automatically¹⁶. The interesting result is that as the number of use cases grows the amount of saving increases. This could be because applications with more use cases tend to have features shared by more use cases. Table 17 shows the savings resulted for the case study *Dilmaj* and Table 18 displays the same results for the case study *AMS*. It can be observed that the savings for the *AMS* case study, which has more use cases, is greater.

Table 17. PIM Elements vs. Automatically Added APSM Elements for Dilmaj Use Cases			
	Number of <i>pim</i> Elements	Number of <i>apsm</i> Elements automatically added	Average Saving
Add Area	16	30	0.65
Add Comment	47	42	0.47
Join Chat	20	47	0.70
Change Sort Criteria	21	19	0.48
Chat	20	44	0.69
Edit Areas	37	57	0.61
Edit Details	31	50	0.62
Edit Use Case	32	39	0.55
Login	18	34	0.65
Logout	12	18	0.60
Privilege	23	47	0.67
Rank	23	47	0.67
Refine Area	24	50	0.68
Register	21	100	0.83

¹⁶ The average is calculated using all results, some of which are not listed here in the thesis.

	Number of <i>pim</i> Elements	Number of <i>apsm</i> Elements automatically added	Average Saving
Search	41	34	0.45
Suggest a Term	42	64	0.60
View Comments	40	50	0.56
View Dilmaj	41	50	0.55
View Homepage	39	43	0.52
View Online Users	16	24	0.60
View Use Case	24	32	0.57
Summary	588	921	0.61

Table 18. PIM Elements vs. Automatically Added APSM Elements for AMS Use Cases

	Number of <i>pim</i> Elements	Number of <i>apsm</i> Elements automatically added	Average Saving
Add Admin	17	40	0.70
Add New Bank	24	62	0.72
Add News	26	78	0.75
Add Tutorial	37	119	0.76
Add User	17	71	0.81
Add Billing Rule	16	46	0.74
Address an Inquiry	31	64	0.67
Apply Penalty	31	121	0.80
Assign Account	35	70	0.67
Cancel Installment	16	59	0.79
Confirm Account Request	22	78	0.78
Confirm Instalment Offer	18	70	0.80
Create Newsletter	24	90	0.79
Edit Billing Rule	26	78	0.75
Edit Personal Information	19	100	0.84
Handle Appointment	26	68	0.72
Inquire a Bill	30	114	0.79
Login	15	39	0.72
Subscribe to Newsletter	19	69	0.78
Unsubscribe from	13	47	0.78

	Number of <i>pim</i> Elements	Number of <i>apsm</i> Elements automatically added	Average Saving
Newsletter			
Pay Online	38	120	0.76
Open Newsletter	29	103	0.78
Print Bills	28	73	0.72
Process Instalment Application	32	125	0.80
Read News	13	24	0.65
Remove Admin	29	119	0.80
Remove Rule	15	52	0.78
Remove Paper/Ebill	19	55	0.74
Apply for Instalments	21	63	0.75
Request an Account	28	144	0.84
Request an Appointment	29	82	0.74
Search Tutorial	30	68	0.69
View Bill	15	28	0.65
View Billing Formula	13	29	0.69
View Bills	15	95	0.86
View Tutorial	42	125	0.75
Summary	858	2788	0.76

Table 19 gives another measure. This table shows the type and the number of elements automatically created at the *apsm* level for every typical element of the *pim*. Other elements are only cloned from *pim* to *apsm*.

Table 19. PIM Elements vs. Automatically Added APSM Elements

PIM	APSM
A submit button (sb1) is grouped with UI components (uis1). The group is contained within a presentation (p1). The presentation represents the UI of a presentation state (ps1) and the submit button (sb1) is related to an sp's outgoing transition (ot1). There are 5+n elements in total, where	1- ps1, p1, uis1, sb1 and ot1 are cloned as ps2, p2, uis2, sb2 and ot2_1. 2- a signal event (se2) is added to ot2_1. For each of the uis1, a parameter (param2) is added to se2 and the type (type2) for param2 is also defined. 3- a state s2_1 is added as the new target of ot2_1.

PIM	APSM
n is the number of elements in <i>uis</i> .	4- an outgoing transition (ot2_2) is added for s2_1. 5- ot2_2 carries a call event (ce2). ce2 calls a controller operation (co2), which in turn calls a service operation (se2). co2 carries a set of parameters like the ones created as param2 of type2 and return a boolean value. 6- a choice (c2) is added with two outgoings, one carrying a guard <i>true</i> and another carrying the guard <i>false</i>. 7- The <i>true</i> outgoing ends at the original target of ot2_1. 8- the <i>false</i> outgoing ends at a presentation state holding an error message. This state has two outgoingings carrying signal events <i>retry</i> and <i>cancel</i>. 1+2n elements are created at step 2, 1 at step 3, 1 at step 4, 3+2n at step 5, 5 at step 6, 7 at step 8, totalling 18+4n.
A use case (uc). 1 element	A use case controller (ucc) is added. 1 element
A data item (d) found in a presentation with n UI components that qualify as data attributes. 1+n element	A data composite, a data entity, attributes to both and their types, a data service, dependencies from service to composite and from composite to entity. 6+2n

Table 19 corresponds to a typical case. The actual numbers may be slightly different depending of the complexity of the *pim* in terms of the number of operations and states. Additionally, some of the elements are re-used. The total number of elements in the PIM is about $7+2n$, n being the number of UI Components. Note that it is assumed that the data item of row 3 is found within the same presentation as the submit button of row 1. The total number of elements created at the APSM level is about $25+6n$. This count does not consider the number of associations that are created between classes at the APSM for primitive parameter/attribute. The ratio of automatically created elements to those manually created is hence, $(25+6n)/(7+2n)$. Even

for large numbers of n , that is large number of UI components within a presentation, this value remains above 3, which means for every PIM element more than 3 elements are in average automatically created. This value remains valid as a global measure throughout different case studies. Table 20 shows the average of savings of the modeling activity with respect to the *APSM* as well as the *AndroMDA*, *GWT*, *WebRatio* and *.Net SPSMs*.

Table 20. Average Savings of the Modeling Activity in Different Platforms

APSM (vs. PIM)	AndroMDA (vs. APSM)	GWT (vs. APSM)	WebRatio (vs. APSM)	.Net (vs. APSM)
75.38%	33.33%	36.78%	46.77%	59.78%

9.7 Summary

In this chapter, the evaluation scenario of the APSM was presented. We first reviewed the criteria set in Chapter 3 in order to verify their coverage by the APSM. By implementing transformations to different platforms and developing applications, we illustrated that the APSM is not dependent upon a specific platform.

A development process, MODEWIS based on the APSM, was introduced. The process was compared to the conventional software engineering process to examine the degree of savings. As Table 19, shows, MODEWIS can eliminate most of the manual parts of modeling activities. It was also discussed that MODEWIS helps maintain the application aligned with requirements by automated mappings between higher-level models and design models.

Finally, results of applying the MODEWIS to actual case studies were presented. Results from the Dilmaj (Table 17) and AMS (Table 18) case studies show that the MODEWIS can be used to save time of development of web applications.

Chapter 10. CONCLUSION

In this thesis, an abstract model for web applications was presented. The volatile nature of the World Wide Web and the rapid change of technologies motivated this thesis. We aimed at a model that could be mapped to more than one platform. In order to obtain such a model, an extensive review of the existing web application models and web development frameworks was performed. Based on the common features of the existing models and emerging technologies, an abstract web model was proposed. From the standpoint of MDA, the abstract web model of this thesis is platform-independent in the sense that it does not depend on any specific web platform. Yet, considering the Web as an abstract platform, it is platform-specific since it identifies an application based on abstract web. Thus, we named this model Abstract-Platform Specific Model (APSM).

A distinguishing aspect of this thesis is the fact that it examines the APSM in the context of a number of transformations targeting customized platforms, referred to as specific platforms. These transformations help assert that the APSM can be mapped to more than one platform. The specific platforms, in this thesis, are defined based on four different technologies, AndroMDA [Andro2007], WebRatio [WebRat2008], Google Web Toolkit [GWT2010] and Microsoft .Net [Net2011]. The transformations were implemented using QVT relations and visualization tools were used to inspect the correctness of the resultant models. We also created case studies and made their models and code available online. In this chapter, we discuss some remaining issues and future work that might improve and add to this thesis.

10.1 Remaining Issues

Some issues remain that may threaten the validity or usability of our approach. Some work has been already started to address such issues. In Section 5.1, we introduced a visual language. This language is built to support a subset of the APSM that can be used to specify the application at the PIM level. However, the initial feedback from developers is that this language suffers from low understandability in its current form. This is a drawback that puts the practical usability of the approach at risk.

Similar to other existing model-driven web development techniques, we created default target platforms for generating executable applications. We even moved forward by adding more than one specific platform. Although this is an advantage, it requires creating new sets of transformations for every specific platform, referred to as platform-specific transformations. Related to this is the fact that specific platforms must also be defined; that is such platforms need to be designed, modelled and tested.

With respect to the implemented mappings, the objective of this thesis was just to demonstrate the possibility of mapping to multiple platforms. But it was also mentioned in Section 9.2 that *apsm-spsm* mappings are expected to be easier to design since the APSM is semantically closer to *spsms* than *pim*. This claim has not been formally validated in this thesis. Additionally, we have not verified how much detail is necessary or enough at the PIM or APSM level. Even with respect to SPSMs, it might be possible to avoid having some details in meta-models and instead create them as part of code generation. It must, however, be noted that code generation is not a focus of our work. We rely on AndroMDA and WebRatio for code generation targeting these respective platforms. We developed utilities for code generation for the GWT and .Net platforms. However, this code is generated only as proof-of-concept. Code generation for these platforms could be done using tools that are specifically designed for such purpose, especially those based on state models such as [Balz2010], [Badr2012] and [UMPL2012].

Several utilities were implemented to support different steps of MODEWIS. But the lack of a unique tool, where all steps of the process could be done in an integrated environment, is another threat to the usability of MODEWIS. Finally, with the constant changes of the World Wide Web models and technologies, the stability of the APSM might be an issue as parts become obsolete or new elements need to be added.

10.2 Future Work

The future work will focus on improving the APSM and its associated transformations, language and process (MODEWIS). This will especially address issues that cause threats to the validity of the approach. Our top priority is the development of a tool that integrates different steps of the MODEWIS process. This tool must be able to integrate the different Eclipse plug-ins and Java utilities created so far.

Developing a user-friendly environment for defining the PIM is also an important future work. This environment will provide a small set of graphical symbols that can be used to create a visual specification of web applications. One possibility is the use of cross-referencing lists that facilitate assigning use cases to states of state machines and triggers to transitions. The ultimate goal would be the development of a user interface that non-experts can understand, and use to define web applications. A working implementation could be made possible using several shortcuts to facilitate the design of the UI and its presentation. For example, as Figure 83 shows, the user does not need to apply the visual language directly for the definition of data operations attached to an element. Instead, drop-down combo boxes can be offered to present existing options and to provide a more user-friendly way of defining the operations.

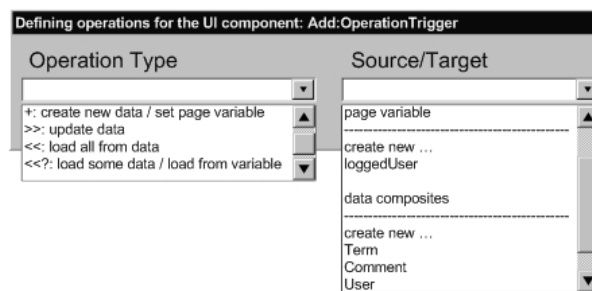


Figure 83 Defining attached operations in practice

Another future work item involves the development of utilities for building specific platforms and transformations. These utilities would be intended for users who want to support additional specific platforms. An option for creating an environment for the definition of transformations is to extending our previous work on the visualization of QVT relations [Fatol2010-2]. A utility for building specific platforms may rely on Eclipse plug-ins.

Extending the APSM to cover upcoming features is another track for future work. We, however, must be careful because expanding the APSM can result in a very complex and large model that may not be maintainable and useful in a longer term. An alternative would be to create different versions of the APSM. While keeping the original APSM intact as possible, we can create more specialized versions for specific areas of the World Wide Web such as Web Services, Web 2.0 and Web Databases. In the same way, it is possible to consider specialized mappings with certain aspects such as security in mind. We have also not discussed the possibility of presenting the APSM as a UML profile or as a DSL because the base model, i.e.

the Botterweck model is neither a UML profile nor a DSL. But it might be beneficial to consider those possibilities, in which case we will need to build the APSM as an independent model first.

A potential piece of future work would be research to establish if the apsm-to-spsm transformations are effectively shorter or easier-to-develop than pim-to-spsm transformations. We can define measures such as lines-of-code, complexity or number-of-relations as basis for the comparison. But a more precise verification requires reviewing the results of research and study for formalizing model transformations such as the ones by Stevens [Steve2007]. A related research question is how to determine the number sufficient and/or necessary details needed at a given level of abstraction. Answering this question will help develop balanced-length transformations. It will require an extensive study of the existing models and their degree of success.

REFERENCES

- [Agra2006] Agrawal, A. Karsai, G. Shi, F. and Vizhanyu, A. The Design of a Language for Model Transformation. *Journal of Software and System Modeling*, Vol 5(3), 2006. pp:261-288
- [Aida2010] Aida/Web Smalltalk, <http://www.aidaweb.si/>, December 2010
- [Akeh2002] Akehurst, D. H. and Kent, S. A Relational Approach to Defining Transformations in a Metamodel. In J.-M. Jézéquel, H. Hussmann, S. Cook (Eds.): *UML 2002 – The Unified Modeling Language 5th International Conference*, 2002, 243-258.
- [Almei2004] Almeida, P. J. Dijkman, R. van Sinderen, M. Pires, L. F.: On the Notion of Abstract Platform in MDA Development. In: *Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2004)*, pp: 253-263.
- [Alter1996] Alter, S. (1996). *Information Systems: A Management Perspective*. Menlo Park, CA: Benjamin Cummings.
- [AMS2010] AMS_APSM, http://www.site.uottawa.ca/~afato092/AMS_APSM_Ali_Fatolahi.zip, December 2010
- [Andro2007] AndromDA, www.andromda.org, 15-02-2007
- [Apach2012] Apache Tomcat, tomcat.apache.org, 15-02-2012
- [Appuk2003] Appukuttan, B. Clark, T. A Model Driven Approach to Model Transformations. In: *Proceedings of the Workshop on Model-Driven Architecture: Foundations and Applications 2003*. pp. 1-12.
- [ASP2010] ASP.Net, www.asp.net, December 2010
- [ATL2005] ATLAS group, ATL: Atlas Transformation Language, ATL Starter's Guide- version 0.1 - December 2005. Available from (<http://www.sciences.univ-nantes.fr/lina/atl/>)
- [Badr2012] Code Generation from Use Cases, www.site.uottawa.ca/~tcl/gradtheses/obadreldin/, April 2012

- [Balz2010] Balz, M. Striewe, M. Goedicke, M. Continuous Maintenance of Multiple Abstraction Levels in Program Code. In: Proceedings of the 2nd International Workshop on Future Trends of Model-Driven Development - FTMDD 2010, Funchal, Portugal, 2010.
- [Baresi2006] Baresi, L. Colazzo, S. Mainetti, L. and S. Morasca. W2000: A Modeling Notation for Complex Web Applications. In: E. Mendes and N. Mosley (eds.) Web Engineering: Theory and Practice of Metrics and Measurement for Web Development. Springer, ISBN: 3-540-28196-7, 2006.
- [Behr2002] Behrens, H. Requirements Analysis Using Statecharts and Generated Scenarios, Doctoral Symposium at IEEE Joint Conference on Requirements Engineering. 2002
- [Beydeda2005] Beydeda, S. Book, M. Gruhn, V. (eds.). Model-Driven Software Development, Berlin; New York : Springer, 2005.
- [Blanc2008] Blanco, C. García-Rodríguez De Guzmán, I. Fernández-Medina, E. Trujillo, J. Piattini, M. Automatic Generation of Secure Multidimensional Code for Data Warehouses: An MDA Approach. In: Proceedings of OTM 2008 pp. 1052 - 1068
- [Blank2004-1] Blankenhorn, K. A UML Profile for GUI Layout, Master's Thesis, University of Applied Sciences Furtwangen, Department of Digital Media, May 23, 2004
- [Blank2004-2] Blankenhorn, K. and Jeckle, M. A UML Profile for GUI Layout. In: Object-Oriented and Internet-Based Technologies, Volume 3263/2004, Springer Berlin / Heidelberg, pp. 110-121.
- [Blog2011] Blogger Widgets, <http://www.bloggerplugins.org/>, March 2011
- [Bogdan2008] Bogdan, C. Falb, J. Kaindl, H. Kavalajian, S. Popp, R. Horacek, H. Arnavotic, E. and Szep, A. Generating an Abstract User Interface from a Discourse Model Inspired by Human Communication. In: Proceedings of HICSS 2008, pp. 36-40.
- [Botter2007] Botterweck, G. 2007. Multi-Front-End-Engineering - Ein modellgetriebener Ansatz zur Entwicklung von Anwendungen mit mehreren Front-Ends, Ph.D. thesis, Koblenz, Germany: Verlag Dietmar Foelbach, ISBN 978-3934795716 <http://www.amazon.de/Multi-Front-End-Engineering-modellgetriebener-Entwicklung-Anwendungen-Fronts-Ends/>
- [Bramb2008] Brambilla, M. Fraternali, P. Tisi, M. A Metamodel Transformation Framework for the Migration of WebML Models to MDA. In: Proceedings of the 4th Int. Workshop on

- Model-Driven Web Engineering (MDWE 2008). N. Koch, G.-J. Houben, A. Vallecillo (Eds.). CEUR Proceedings, volume 389, pp. 91-105.
- [Bray2002] Bray, K. I. An Introduction to Requirements Engineering, Addison Wesley, 2002
- [Caram2004] Caramazana, A. Angel, M. Vidales, S. and Joyanes, L. MDA Model Transformation using UML Collaborations. In: Proceedings of the International Computer Engineering Conference 2004.
- [Cácer2003] Cáceres, P., Marcos, E., Vela, B. A MDA-Based Approach for Web Information System Development. In: Proceedings of Workshop in Software Model Engineering (WiSME), Proceedings of UML 2003.
- [Catal2010] Catalyst Web Framework, <http://www.aidaweb.si/>, December 2010
- [Ceri2000] Ceri, S. Fraternali, P. and Bongio, A. Web Modeling Language (WebML): a modeling language for designing Web sites. Computer Networks 33 (1-6): 137-157 (2000)
- [Ceri2006] Ceri, S. Fraternali, P. Bongio, A. Brambilla, M. Comai, S. Matera, M. Designing Data-Intensive Web Applications. Morgan Kaufmann. 2006
- [Chaos2001] The Standish Group International Report, Extreme Chaos, 2001.
- [Cich2007] Cichetti, A. Di Ruscio, D. Di Salle, A. Software customization in model driven development of web applications. In: Proceedings of SAC 2007. pp. 1025-1030
- [Cich2008] Cichetti, A. Di Ruscio, D. Decoupling Web Application Concerns through Weaving Operations. Science of Computer Programming 70(1) 2008. pp. 62-86
- [Coc2001] Cockburn, A. Writing Effective Use Cases. Boston: Addison-Wesley, c2001.
- [Cold2010] ColdFusion, <http://cfwheels.org/>, December 2010
- [Cost2006] Costa, D. Nóbrega L. and Nunes, N. J. An MDA Approach for Generating Web Interfaces with UML ConcurTaskTrees and Canonical Abstract Prototypes, 137-152. In Book: Task Models and Diagrams for Users Interface Design
- [CPPCMS2010] CPPCMS, <http://cppcms.sourceforge.net/wiki/en/page/main>, December 2010
- [DaSil2003] Da Silva P. P. and Paton, N. W. Improving UML Support for User Interface Design: A Metric Assessment of UMLi. In: Proceedings of the ICSE Workshop on SE-HCI 2003: pp. 76-83

- [Demeur2005] 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.
- [DeSouz2008] De Souza, R. A. C. de Barros, R. S. M. A Model-Driven Method for the Development of Web Applications User Interaction Layer. In: Proceedings of TASE 2008: pp. 91-98
- [DeTroy1998] De Troyer, O. Leune, C. WSDM: A User-Centered Design Method for Web Sites. In: Proceedings of Computer Networks and ISDN Systems. In: 7th International World Wide Web Conference. 1998. pp. 85–94.
- [Diam2008] Diamodl, <http://www.idi.ntnu.no/~hal/research/diamodl>, July 19, 2008
- [Dijk1982] Dijkstra, Edsger W. (1982), "On the role of scientific thought", in Dijkstra, Edsger W., Selected writings on Computing: A Personal Perspective, New York, NY, USA: Springer-Verlag New York, Inc., pp. 60–66, ISBN 0-387-90652-5, <http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html>
- [Dilmaj2010] Dilmaj, http://sokhangozaar.appspot.com/?locale=en_US#, July 2010
- [Dilmaj2011] Dilmaj, http://dilmajtest.appspot.com/?locale=en_US#, July 2011
- [Dist2007] Distant, D. Pedone, P., Rossi, G. and Canfora, G. Model-Driven Development of Web Applications with UWA, MVC and Java Server Faces. In: Web Engineering. Volume 4607/2007. Springer Berlin / Heidelberg. Pages 457-472
- [EMF2009] Eclipse Modeling - EMF - Home, www.eclipse.org/modeling/emf, January 2009
- [Enges1987] Engeström, Y. (1987). Learning by Expanding: An Activity-Theoretical Approach to Developmental Research. Helsinki: Orienta-Konsultit.
- [Epner2000] 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>
- [Escal2003] Escalona, M.J. Torres, J. Mejias, 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

- [Escal2008] 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
- [Escal2009] Escalona, M.J. Morero, F. Parra, C.L. Nieto, J. Pérez, F. Martín, F. Llergo A. and Gutiérrez J.J. A Practical Environment to Apply Model Driven Web Engineering. In: Information Systems Development. Vol. 1. 2009. pp. 249-257
- [Face2011] App Directory | Facebook, <http://www.facebook.com/apps/directory.php>, March 2011
- [Fatol2008-1] Fatolahi, A. Somé S. S. and Lethbridge, T. C. A Model-Driven Approach for the Semi-Automated Generation of Web-based Applications from Requirements. In: Proceedings of International Conference on Software Engineering and Knowledge Engineering 2008. pp. 619-624.
- [Fatol2008-2] Fatolahi, A. Somé S. S. and Lethbridge, T. C. Towards A Semi-Automated Model-Driven Method for the Generation of Web-based Applications from Use Cases. In: Proceedings of the 4th Model Driven Web Engineering Workshop (MDWE2008). MoDELS'2008. pp. 31-45.
- [Fatol2008-3] Fatolahi, A. Somé S. S. and Lethbridge, T. C. Automated Generation of Abstract Web Applications using QVT Relations, August 2008, Technical Report TR-2008-02, School of Information Technology and Engineering, University of Ottawa. Available from: www.site.uottawa.ca/eng/school/publications/techrep/2008
- [Fatol2010-1] Fatolahi, A. Somé S. S. and Lethbridge, T. C. Automated Generation of Abstract Web Models using QVT Relations, September 2010, Technical Report TR-2010-06, School of Information Technology and Engineering, University of Ottawa. Available from: www.site.uottawa.ca/eng/school/publications/techrep/2010
- [Fatol2010-2] Fatolahi, A. Somé S. S. and Lethbridge, T. C. Designing a Map of Mappings: Visualization of QVT Relations using Basic Petri-Nets. In: Proceedings of the 2nd International Workshop on Future Trends of Model-Driven Development, 12th International Conference on Enterprise Information Systems, 8 - 12 June, 2010, Funchal, Madeira – Portugal. pp. 33-45.
- [Fatol2010-3] Fatolahi, A. An Integrated Visual Language for Modeling Web UIs. Available from: modewis.blogspot.com/2010/08/integrated-visual-language-for-modeling.html

- [Fatol2010-4] Fatolahi, A. Automated Generation of Abstract Web Models using QVT Relations. Available from: modewis.blogspot.com/2010/08/automated-generation-of-abstract-web.html
- [Fatol2010-5] Fatolahi, A. Somé S. S. and Lethbridge, T. C. A Meta-Model for Model-Driven Development of Web Applications. Technical Report, July 2010, Available from: www.site.uottawa.ca/eng/school/publications/techrep/2010/TR-2010-04.pdf
- [Fatol2011-1] Fatolahi, A. Somé S. S. and Lethbridge, T. C. Model-Driven Web Development for Multiple Platforms. Journal of Web Engineering, Vol. 10 (2011) No. 2. (pp109-152). June 2011.
- [Fatol2011-2] Fatolahi, A. Somé S. S. and Lethbridge, T. C. Towards Reusability In Web Modeling Using QVT Relations. In: Proceedings of the 7th International Conference of Web Information Systems and Technologies (WEBIST 2011), pp. 383-386.
- [Fatol2012] Fatolahi, A. Somé S. S. and Lethbridge, T. C. A Meta-Model for Model-Driven Web Development. International Journal of Software and Informatics, Special Issue on Software Modeling, Volume 6, Issue 2 (2012), pp. 125-162.
- [Flash2010] Adobe Flash rich internet applications, <http://www.adobe.com/products/flashplayer/>, 2010-01-03
- [Flash2011] Flash Developer Centre, <http://www.adobe.com/devnet/flash.html>, March 2011
- [Fons2003] Fons, J. Valderas, P. Ruiz, M. Rojas, G. Pastor, O. OOWS: A Method to Develop Web Applications from Web-Oriented Conceptual Models. In: Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI).
- [Frank2003-1] Frankel, D. Model Driven Architecture: Applying MDA to Enterprise Computing. New York: Wiley, c2003.
- [Frank2003-2] Frankel, D. S. Harmon, P. Mukerji, J. Odell, J. Owen, M. Rivitt, P. Rosen, M. Soley, R. M. The Zachman Framework and the OMG's Model Driven Architecture, a BP Trends Whitepaper, September 2003 (Available from: http://www.omg.org/mda/mda_files/09-03-WP_Mapping_MDA_to_Zachman_Framework1.pdf)

- [Freud2008] Freudenstein, P. Nussbaumer, M. Allerdig, F. Gaedke, M. A Domain-Specific Language for the Model-Driven Construction of Advanced Web-Based Dialogs. In: Proceedings of the 17th international conference on World Wide Web, pp. 1069-1070, 2008.
- [Fuse2010] Fusebox, <http://www.fusebox.org/>, December 2010
- [GAE2011] Google Application Engine – Google Code, <http://code.google.com/appengine/>, March 2011
- [Galit2007] Galitz, W. O. The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques. Indianapolis, Wiley Pub., c2007.
- [Garr2007] Jesse James Garrett, Ajax: A New Approach to Web Applications, Available from: <http://www.adaptivepath.com/ideas/essays/archives/000385.php>. Retrieved 2007-03-05
- [Gomez2000] Gómez, J., Cachero, C. and Pastor, O. Extending an object-oriented conceptual modelling approach to Web application design. In: Benkt Wangler and Lars Bergman, (Eds.). Proceedings of the 12th International Conference on Advanced Information Systems Engineering, pp. 79-93.
- [Greer2008] Derek Greer <http://www.ctrl-shift-b.com/2008/01/art-of-separation-of-concerns.html>
- [GWT2010] Google Web Toolkit, Google Code, <http://code.google.com/webtoolkit/>, June 2010
- [Hamm2008] Hammoudi, S. Alouini, W. Lopes, D. Towards a Semi-Automatic Transformation Process. In: MDA - Architecture and Methodology. In: International Conference of Enterprise Information Ssystems (3-2) 2008, pp. 416-425/
- [Han2007] Han, W. M. and Huang, S. J. An empirical analysis of risk components and performance on software projects, Journal of Systems and Software, Vol. 80 (1) pp. 42-50, 2007.
- [He2005] 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

- [Hern2010] Hernández, C. Quintero, R. and Sánchez, L. Z.: Towards the Definition of a Metamodel for the Conceptual Specification of Web Applications Based on Social Networks. ICCSA (2) 2010, pp. 361-369.
- [Hruby2006] Hruby, P. Model-Driven Design using Business Patterns; with contributions by Jesper Kiehn and Christian Vibe Scheller. Berlin ; New York : Springer-Verlag, c2006.
- [HTML2010] HTML5, <http://www.w3.org/TR/html5/>, 2010-05-05
- [Jaim2011] Carlos R. Jaimez-González, Christian Sánchez-Sánchez, J. Sergio Zepeda-Hernández, A Web Platform for Creating and Administering Interactive Online Tutorials, In: Proceedings of International Canada Education Conference 2011, pp. 88-92.
- [JDBC2010] JDBC Overview, <http://java.sun.com/products/jdbc/overview.html>, July 2010
- [JDO2010] Java Data Objects, <http://java.sun.com/jdo/>, July 2010
- [JMI2008] java.sun.com/products/jmi, Feb 13, 2008
- [Joom2001] Joomla!, <http://www.joomla.org>, March 2011
- [JSF2010] JavaServer Faces Technologies, www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html, Feb, 1st, 2010
- [JSON2012] JSON, json.org, Feb, 1st, 2012
- [JSP2008] Java Server Pages Technology, www.oracle.com/technetwork/java/javaee/jsp, Jun, 15th, 2008
- [Kappel2008] Kappel, G. Kargl, H. Reiter, T. Retschitzegger, W. Schwinger, W. Strommer, M. and Wimmer, M. A Framework for Building Mapping Operators Resolving Structural Heterogeneities. In: Proceedings of the Information Systems and e-Business Technologies (UNISCON'2008), Springer, pp. 158-174.
- [Kasc2004] Kasehk, R. Schewe K. D. Wallace, C. and Matthews, C. Story Boarding for Web-Based Information Systems. In: Taniar, D. and Rahayu J. W. Web Information Systems, pp. 1-33, IGI Global, 2004.
- [Kater2008] 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

- [Kaval2007] Kavaldjian, S. A Model-Driven Approach to Generating User Interfaces. In: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, Doctoral symposiumPages, pp. 603 – 606, 2007
- [Klep2003] Kleppe, J. Warmer, W. Bast. MDA Explained, The Model-Driven Architecture: Practice and Promise, Addison Wesley, 2003.
- [Koch2000] 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.
- [Koch2006] 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
- [Kroiß2008] Kroiß, C. and Koch, N. UWE Metamodel and Profile User Guide and Reference, Technical Report, February 2008. Available from (www.pst.ifi.lmu.de/projekte/uwe)
- [Kusel2009] Kusel, A. Schwinger, W. Wimmer, M. Retschitzegger, W. Common Pitfalls of Using QVT Relations - Graphical Debugging as Remedy. In: Proc. 2009 14th IEEE International Conference on Engineering of Complex Computer Systems. pp. 329-334
- [Lara2009] de Lara, J. and Guerra, E. Formal Support for QVT-Relations with Coloured Petri Nets. Model Driven Engineering Languages and Systems (2009), pp. 256-270.
- [Larman2005] Larman, C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Prentice Hall PTR. 2005.
- [Laues2005] Lauesen, S. User Interface Design: A Software Engineering Perspective. Harlow, UK ; New York : Pearson/Addison-Wesley, 2005.
- [Lee1998] 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. 1998. pp. 121–38
- [Leth2001] Lethbridge T. C., Laganière R., Object-Oriented Software Engineering: Practical Software Development using UML and Java, London : McGraw-Hill, 2001

- [Li2000] 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. TOOLS - Asia 2000. pp. 265-274
- [Lian2007] 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 (ICSOF 2007), INSTICC Press. pp. 109 – 116.
- [Mars2003] Marschall F. and Braun, P. Model Transformations for the MDA with BOTL. In: A. Rensink (Ed.) Proceedings of the Workshop on Model Driven Architecture: Foundations and Applications, 2003, pp. 25-36.
- [Mazón2006] Mazón, J-N. Trujillo, J. and Lechtenbörger, J. A Set of QVT Relations to Assure the Correctness of Data Warehouses by Using Multidimensional Normal Forms. Conceptual Modeling - ER 2006. pp. 385-398.
- [McNei2004] McNeile, A. and Simons, N. Methods of Behaviour Modelling: A Commentary on Behaviour Modelling Techniques for MDA. White Paper (Draft). Metamaxim Ltd, 2004.
- [MDWE2011] Model-Driven Web Engineering Workshops, www.wikicfp.com/cfp/servlet/event.showcfp?eventid=15825©ownerid=2, 24 April 2011
- [MDWSGWT2010] MODEWIS_GWT_Generator, September 2010, site.uottawa.ca/~afato092/modewis_gwt_generator.jar
- [MDWSQVT2011] MODEWIS QVT Relations, October 2011, site.uottawa.ca/~afato092/MODEWIS_Transformations.zip
- [MDWSPR2010] MODEWIS_ProofRead, site.uottawa.ca/~afato092/modewis-proofread.vsd, September 2010
- [MDA2003] OMG, MDA Guide Version 1.0.1, 12th June 2003
- [Medin2008] mediniQVT Trac, <http://projects.ikv.de/qvt>, 3 May 2008
- [Meliá2005-1] Meliá, S. Gómez, J. and Koch, N. Improving Web Design Methods with Architecture Modeling. In: Proceedings of the 6th International Conference on Electronic

- Commerce and Web Technologies (EC-Web 2005). In: Kurt Bauknecht, Birgit Pröll, Hannes Werthner (Eds.). LNCS 3590, Springer Verlag, pp. 53-64, August 2005.
- [Meliá2005-2] Meliá, S. Gómez, J. Applying Transformations to Model Driven Development of Web Applications. ER (Workshops) 2005, pp. 63-73
- [Meliá2006] Meliá, S., Gomez, J. UPT. A Graphical Transformation Language based on a UML Profile. In: Proceedings of European Workshop on Milestones, Models and Mappings for Model-Driven Architecture (3M4MDA 2006). 2nd European Conference on Model Driven Architecture (EC-MDA 2006) (July 2006)
- [Mellor2004] Mellor, S. J. Scott, K. UHL, A. Weise, D. MDA Distilled: Principles of Model-Driven Architecture. Boston: Addison-Wesley, 2004.
- [Mendes2006] Mendes, E. and Mosley, N. S. *Web Engineering*, Berlin; New York : Springer, 2006.
- [MOF2007] OMG's MetaObject Facility (MOF) Homepage, <http://www.omg.org/mof/>, February 15th, 2007
- [Molin2008] Molina, F. Pardillo, J. Ambrosio, J. and Álvarez, T. Modelling Web-Based Systems Requirements Using WRM. In: Proceedings of WISE Workshops 2008. pp. 122-131
- [Moment2008] MOMENT-QVT, <http://moment.dsic.upv.es/content/view/34/75/>, May 10, 2008
- [Mood2011] Moodle, <http://moodle.org/>, 15 April 2011
- [Moren2006] 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.
- [Moren2007] Moreno, N. Fraternali, P. Vallecillo, A. WebML modeling in UML. IET Software Journal (2007).
- [MSQL2012] MySQL, The world's most popular open source database, www.mysql.com, April 6, 2012.
- [Mull2005] Muller, P. A. Studer, P. Fondement, F. and Bézivin, J. Platform-Independent Web Application Modeling and Development with Netsilon. Software and System Modeling 4(4), pp. 424-442 (2005)
- [MVC2011] ASP.NET MVC, www.asp.net/mvc, October 30, 2011

- [Net2011] Microsoft .Net framework, <http://www.microsoft.com/net/>, February 2011
- [NetCraft2011] Web Server Survey Archives – Netcraft, news.netcraft.com/archives/web_server_survey.html, March 22, 2011
- [Nguy2006] Nguyen, P. Chun, R. Model Driven Development with Interactive Use Cases and UML Models. Software Engineering Research and Practice 2006. pp. 534-540
- [Nikol2005] Nikolaidou, M. and Anagnostopoulos, D. A Systematic Approach for Configuring Web-Based Information Systems. Journal of Distributed and Parallel Databases, Springer Netherlands, Volume 17, Number 3 / May, 2005, pp. 267-290
- [Nune2006] Nunes, D. A. and Schwabe, D. Rapid Prototyping of Web Applications Combining Domain Specific Languages and Model Driven Design. In: Proceedings of ICWE 2006, pp.153-160
- [OOH4RIA2011] OOH4RIA, suma2.dlsi.ua.es/ooh4ria, Jan 6th, 2011
- [OOHDM2007] The Object-Oriented Hypermedia Design Method (OOHDM) <http://www.oohdm.telemidia.puc-rio.br/oohdm.html>, July 2007.
- [Past2007] Pastor O. and Molina J. C. Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling, Springer (2007)
- [Patra2004] Patrascioiu, O. YATL: Yet Another Transformation Language. In: Proceedings of Model-Driven Architecture with Emphasis on Industrial Applications Workshop 2004.
- [Peter1981] Peterson, J. L. Theory and the Modeling of Systems. Prentice-Hall, N.J., 1981.
- [Phalp2007] Phalp, K.T., Vincent, J.V and Cox, K., Assessing the Quality of Use Case Descriptions, Software Quality Journal, vol. 15 (1), pp. 69-97. 2007
- [PHP2012] PHP, Hypertext Preprocessor, April 6, 2012, www.php.net
- [Pier2007] Pierantonio, A. Vallecillo, A. Selic, B. and Gray, J. Special Issue on Model Transformation. Sci. Comput. Program. 68(3): 111-113 (2007)
- [PIM2010] MODEWIS_PIM_Generator, September 2010, site.uottawa.ca/~afato092/modewis_pim_generator.jar
- [Pu2008] Pu, J. Yang, H. Xu, B. Xu, L. Chu, W.C.C. Combining MDE and UML to Reverse Engineer Web-Based Legacy Systems. In: Proceedings of COMPSAC 2008. pp. 718-725

- [QVT2008] OMG, MOF QVT Specification, www.omg.org/spec/QVT/1.0/PDF/, 2008-04-03
- [Reen1979] Trygve Reenskaug: MODELS - VIEWS - CONTROLLERS. Technical note, Xerox PARC, December 1979. A scanned version on <http://heim.ifi.uio.no/~trygver/mvc/index.html>
- [Reit2007] Reiter, T. Wimmer, M. and Kargl, H. Towards a runtime model based on Colored Petri Nets for the Execution of Model Transformations. In: Proc. 3rd Workshop on Models and Aspects. ECOOP'07, Berlin.
- [Romei2008] Romeikat, R., Roser, S., Müllender, P., Bauer, B. "Translation of QVT relations into QVT operational mappings" In the proceedings of the 1st International Conference on Model Transformations, ICMT 2008. pp. 137-151
- [Rosen2007] Rosenberg, D. and Stephens, M. Use Case Driven Object Modeling with UML : Theory and Practice Berkeley, CA : Apress ; New York , 2007.
- [Ross2006] Rossi, G. Schwabe, D. Model-Based Web Application Development. 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.
- [Ruby2010] Ruby on rails, www.rubyonrails.org, December 2010
- [RUP2009] IBM – Rational Unified Process (RUP), <http://www-01.ibm.com/software/awdtools/rup/>, September 2009
- [Sack2007] 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
- [Schat2005] Schattkowsky, T. and Lohmann, M. Towards Employing UML Model Mappings for Platform Independent User Interface Design. Springer Berlin / Heidelberg. Volume 3844/2006, Satellite Events at the MoDELS 2005 Conference, pp. 201-209
- [Schauer2007] Schauerhuber, A. Wimmer, M. Kapsammer, E. Schwinger, W. Retschitzegger W. (2007). Bridging WebML to Model-Driven Engineering: from Document Type Definitions to Meta Object Facility. IET SOFTWARE, 1-3, pp. 81 - 97
- [Scheid2006] Scheidgen, M. Model Patterns for Model Transformations in Model Driven Development. In: Proceedings of the Fourth Workshop on Model-Based Development of

- Computer-Based Systems and Third International Workshop on Model-Based Methodologies for Pervasive and Embedded Software, pp. 149 – 158. 2006
- [Schmid2000] Schmidt, D. Stal, M. Rohnert, H. and Buschmann, F. Pattern-Oriented Software Architecture. Chichester [England]; New York: Wiley, <2000-c2007>
- [Schmid2005] Schmid, H. A. and Donnerhak, O. The PIM to Servlet-Based PSM Transformation with OOHMDA. In: Proceedings of Workshop on Model-driven Web Engineering (MDWE 2005)
- [Seam2010] Seam Framework, <http://seamframework.org/>, December 2010
- [Selic2003] Selic, B. The Pragmatics of Model-Driven Development. IEEE Software, v.20 n.5, p.19-25, September 2003.
- [Selic2006] Selic B. Model-Driven Development: Its Essence and Opportunities. Ninth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2006, pp 313-319.
- [Sokhan2010] Sokhangozaar, <http://code.google.com/p/sokhangozaar/>, January 2010
- [Sokhan2011] Sokhangozaar, <https://github.com/sohale/dilmajtemp>, January 2011
- [Shklar2003] Shklar, L. Rosen, R. Web Application Architecture: Principles, Protocols, and Practices. Chichester, England ; Hoboken, NJ : John Wiley, 2003.
- [Smart2008] SmartQVT, <http://smartqvt.elibel.tm.fr/>, 5 May 2008
- [Somé2006] Somé, S. S. Supporting Use Case Based Requirements Engineering. Information and Software Technology Vol. 48 (1), pp. 43-58. 2006.
- [Somé2008] Stéphane Sotèg Somé's homepage, <http://www.site.uottawa.ca/~ssome/>, Fall 2008.
- [Sous2007] 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.
- [Sprin2008] Spring Framework, <http://www.springframework.org/>
- [Stocq2004] Stocq, J. and Vanderdonck, J. A Domain Model-Driven Approach for Producing User Interfaces to Multi-Platform Information Systems. In: Proceedings of the Working Conference on Advanced Visual Interfaces, 2004. pp. 395-398

- [Strut2010] Apache Struts, <http://struts.apache.org/>, December 2010
- [Stahl2006] 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)
- [Steph2001] Stephanidis, C. User Interfaces for all Concepts, Methods, And Tools, Mahwah, N.J.: Lawrence Erlbaum Associates Inc., 2001.
- [Steve2007] Stevens, P. A Landscape of Bidirectional Model Transformations, Generative and Transformational Techniques in Software Engineering II: International Summer School, GTTSE 2007, pp. 408 – 424
- [Strut2011] <http://struts.apache.org/>, Feb. 04, 2011
- [Suh2001] Suh, W. Lee, H. A Methodology for Building Content-Oriented Hypermedia Systems. Journal of Syst Software 2001, 56. pp. 115–31.
- [Suka2007] Sukaviriya, N. Sinha, V. Ramachandra, T. Mani, S. Model-Driven Approach for Managing Human Interface Design Life Cycle. MoDELS 2007. pp. 226-240
- [Sut1998] Sutcliffe, A. G. Maiden, N. A. M., Minocha, S. Manuel, D. Supporting Scenario-Based Requirements Engineering. IEEE Transactions on Software Engineering, Vol. 24 (12) pp. 1072-1088, 1998.
- [Taleb2007] Taleb, M. Seffah, A. Abran, A. Model-Driven Architecture for Web Applications. Human-Computer Interaction (1) 2007. pp. 1198-1205
- [O'R2005] 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. Retrieved 2006-08-06.
- [Tong2003] Tongrungrrojana, R. Lowe, D. WebML+: A Web Modeling Language for Forming a Bridge Between Business Modeling and Information Modeling. In: Proceedings of SEKE 2003. pp. 17-24.
- [Tong2004] Tongrungrrojana, R. Lowe, D. WIED: A Web Modelling Language for Modelling Architectural-Level Information Flows. J. Digit. Inf. 5(2): 2004
- [UCEd2005] Use Case Editor, site.uottawa.ca/~ssome/Use_Case_Editor_UCEd.html, Dec. 15, 2005

- [UCM2008] UCMs, <http://www.usecasemaps.org>, July 18, 2008
- [Uden2008] Uden, L, Valderas, P. and Pastor, O. An Activity-Theory-Based Model to Analyse Web Application Requirements. Journal of Information Research, Vol. 13, No. 2, June 2008.
- [UML2007] OMG, Unified Modeling Language: Superstructure, February 2007
- [UMPL2012] Umple User Manual: Basic State Machine, <http://statemachines.umple.org/>, April 2012
- [UWA2002] UWA Consortium, Ubiquitous Web Applications. In: Proceedings of the eBusiness and eWork Conference 2002, (e2002: October 16-18 2002, Prague, Czech Republic) (2002)
- [Valv2009] Valverde, F. and Pastor, O. Facing the Technological Challenges of Web 2.0: A RIA Model-Driven Engineering Approach. In: Proceedings of WISE 2009: pp. 131-144
- [Vander2005] Vanderdonckt, J. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In: Advanced Information Systems Engineering, Volume 3520/2005, Springer Berlin / Heidelberg, pp. 16-31.
- [Vara2007] 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
- [Varró2006] Varró, D. Model Transformation By Example. In: Proceedings of the 9th International Conference, MoDELS 2006, pp. 410-424,
- [Varró2007] Varró, D. Balogh, Z. Automating Model Transformation by Example Using Inductive Logic Programming. In: Proceedings of SAC 2007, pp. 978-984
- [Vdov2005] Vdovjak, R. and Houben, G. J. A Model-Driven Approach for Designing Distributed Web Information Systems. Web Engineering, 2005. pp. 453-464
- [Vern2005] Verner, J. Cox, K. Bleistein, S. J. and Cerpa, N. Requirements Engineering And Software Project Success: An Industrial Survey In Australia And The US, Australian Journal of Information Systems, vol. 13, pp. 225-238, 2005.
- [Vis2010] Visio Information, visio.mvps.org, March 2010
- [VisWeb2011] Microsoft Web Platform, <http://www.microsoft.com/express/Web/>, March 2011

- [VMTS2009] VMTS, <http://avalon.aut.bme.hu/~tihamer/research/vmts/>, Decemebr 2009
- [Wagner2006] Wagner, F. Schmuki, R. Wagner T. and Wolstenholme, P. Modeling Software With Finite State Machines: A Practical Approach. Ferdinand Boca Raton, FL: Auerbach Pub., 2006.
- [Wall2011] Wallis, M. Henskens, F. and Hannaford, M. Classification And Modelling Of Web Technologies. In: Proceedings of the 7th International Conference of Web Information Systems and Technologies (WEBIST 2011), pp. 151-156.
- [Wan1995] Wan, J. Bieber, M. P. Wang, J. T. L. Ng, P. A.: LHM: A Logic-Based Hypertext Data Model For Integrating Hypertext And Information Systems. In: Proceedings of HICSS (3) 1995, pp. 350-354
- [Wan1996] Wan, J. Bieber, M.: GHMI: A General Hypertext Data Model Supporting Integration of Hypertext and Information Systems. In: Proceedings of HICSS (2) 1996, pp. 47-55
- [Wiki2011] Wikimedia Foundation, <http://www.wikimedia.org/>, 15 April 2011
- [Web2011] Web Engineering, <http://webengineering.org/>, March 2011
- [WebCT2011] Blackboard Home, <http://www.blackboard.com/>, 15 April 2011
- [WebRat2008] WebRatio, www.webratio.com, 2-2-2008
- [WebRat2011] Meta-Model, WebRatio Forum, groups.google.com/group/webratio/browse_thread/thread/1d5fe8425ea77da1?hl=en, 15 April 2011
- [WebTE2008] WebTE, WebSA Transformation Engine, <http://ebusiness.dlsi.ua.es:8080/WebSATool/>, July 11 2008
- [Weid1998] Weidenhaupt, K. Pohl, K. Jake, M. Haumer, P. Scenarios In System Development: Current Practice. IEEE Software 1998;2. pp. 34–45.
- [Whit2004] Whitehead Jr. E. J., Ge, G. Pan, K. Automatic Generation Of Hypertext System Repositories: A Model Driven Approach. Hypertext 2004, pp. 205-214
- [Will2003] E. D. Willink. UMLX: A Graphical Transformation Language For MDA. In A. Rensink, editor, TR-CTIT-03-27, pp. 13-24, Enschede, 2003.

- [Wimm2009-1] Wimmer, M. Kappel, G. Schönböck, J. Kusel, A. Retschitzegger, W. Schwinger, W. A Petri Net based Debugging Environment for QVT Relations. In: Proc 24th International Conference on Automated Software Engineering (ASE 2009), IEEE, pp. 1-12, 2009
- [Wimm2009-2] Wimmer, M. Kusel, A. Schoenboeck, J. Kappel, G. Retschitzegger, W. and Schwinger, W. Reviving QVT Relations: Model-Based Debugging Using Colored Petri Nets. In: Proc 12th Int. conference on Model Driven Engineering Languages and Systems (2009) pp. 727-732.
- [Wimm2009-3] Wimmer, M. Kusel, A. Reiter, T. Retschitzegger, W. Schwinger, W. Kappel, G. Lost in Translation? Transformation Nets to the Rescue! Information Systems: Modeling, Development, and Integration (2009), pp. 315-327.
- [Wimm2009-4] Wimmer, M. Kusel, A. Schönböck, J. Reiter, T. Retschitzegger, W. Schwinger, W. Lets's Play the Token Game - Model Transformations Powered By Transformation Nets. Proc. of the International Workshop on Petri Nets and Software Engineering PNSE'09, Université Paris 13, (2009), S. 35 - 50.
- [WKSH2010] MODEWIS_yEd_Workshop,
http://www.site.uottawa.ca/~afato092/MODEWIS_yEd_Workshop.zip, Dec 2010
- [WS2011] Web Services and Service-Oriented Architecture, <http://www.service-architecture.com>, Dec 2011.
- [Wu2007] 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
- [XMI2007] OMG, MOF 2.0/XMI Mapping, Version 2.1.1, December 2007
- [XSLT2011] XSL Transformation (XSLT), <http://www.w3.org/TR/xslt>, December 2011.
- [YED2010] Yed graph Editor, http://www.yworks.com/en/products_yed_about.html, February 2010

Appendix A. CLASS DESCRIPTIONS

The complete description of classes of the APSM is presented in this appendix using the pattern of the Listing 18:

Listing 18 The Pattern of Class Descriptions

n ClassName (from PackageName)

n indicates the order of the class according to the sections of this appendix. ClassName is the name of the class and PackageName is the package this class is borrowed from. If the package is a UML packages used in the Botterweck model, the UML package is mentioned for better understandability. Additional descriptions might appear in this section.

Generalizations

Indicates which class(es) this class inherit from. If there is no generalization, this section is omitted.

Semantics

This describes the semantics of the class as used in the APSM, which may be more specific than the one the original model.

Attributes

This is the list of the attributes that are not inherited from other classes, if the class does not introduce any new attributes, this section is omitted.

Operations

This is the list of the operations that are not inherited from other classes, if the class does not introduce any new operations, this section is omitted.

Associations

This is the list of the associations that are not inherited from other classes, if the class does not introduce any new associations, this section is omitted. As an exception, this section might include dependencies as well.

Constraints

This is the list of the constraints that are not inherited from other classes and are different from what the attributes, associations and their properties impose on the class; if the class does not introduce any new attributes, this section is omitted. Constraints are either at the PIM or the APSM level. Constraints of the APSM level are supported by the QVT relations that automatically create the APSM and ensure the constraints are guarded. Constraints of the PIM level are modeled using OCL. In the latter case the code of the constraint is presented here.

Presentation

If the visual language – Section 4.5 - supports this class, the notation or the equivalent physical form is mentioned here. If there is no presentation, this section is omitted.

Extensions

This is the list of alterations caused by this class to the Botterweck model. If there is no extensions, this section is omitted.

A.1 Account (from APSM)

Generalizations

DataEntity.

Semantics

This represents an online account, which needs authorization to be accessed and is likely to change the view of the application.

Attributes

- id, the unique identifier of the object determined automatically by the database.
- username, the username associated with this account.
- password, the password associated with this account.
- email, the email associated with this account
- status, this can be any of the enumerations defined in the enumeration *AccountStatus*.

A.2 AccountComposite (from APSM)

Generalizations

DataComposite.

Semantics

This represents an online account, which needs authorization to be accessed and is likely to change the view of the application. This is the composite version of the class *Account* that is used to transfer information between layers.

Attributes

- id, the unique identifier of the object determined automatically by the database.
- username, the username associated with this account.
- password, the password associated with this account.
- email, the email associated with this account
- status, this can be any of the enumerations defined in the enumeration *AccountStatus*.

Associations

- Dependency to *Account*.
 - *member: Member[1]* is the member associated with this account.

A.3 AccountService (from APSM)

Generalizations

Service.

Semantics

This represents an online account, which needs authorization to be accessed and is likely to change the view of the application. This is the composite version of the class *Account* that is used to transfer information between layers.

Attributes

- id, the unique identifier of the object determined automatically by the database.
- username, the username associated with this account.
- password, the password associated with this account.
- email, the email associated with this account
- status, this can be any of the enumerations defined in the enumeration *AccountStatus*.

Associations

- Dependency to *Account*.
 - *member: Member[1]* is the member associated with this account.

A.4 AccountStatus (from APSM)

Semantics

This is an enumeration containing all possible states of an account.

Attributes

It has the following enumeration literals:

- undefined (=0), account just created or is missing key information.
- locked (=1), account is locked due to suspicious activities such as too many unsuccessful login attempts.
- active (=2), account is working fine.

- awaiting (=3), account is waiting to be activated. This happens when an application expects the User to activate her account in a second step by clicking on a link for example.

A.5 ActionType (from APSM)

Semantics

This is an Enumeration that carries the default actions that may be associated with UI components.

Attributes

Enumerations are as follows:

- undefined (=0), for exceptions when the type of action may not be identified.
- email (=1), operations of this type email the contents of the source UI component to the email address specified in the target UI component.
- isEqual (=2), this action is a conditional action that checks the contents of two UI components. The successful matching of the two components will be considered as the precondition of leaving the current presentation state.
- set (=3), sets the value of a UI Component.
- resent (=4), resets the value of a UI Component.
- populate (=5), populates the contents of a UI composite.

Presentation

At the visual level, each action type is identified using a specific symbol:

- Email: @
- isEqual: =

Extensions

The Botterweck model is extended by adding the enumeration, ActionType to it.

A.6 ActivateAccount (from APSM)

Generalizations

UseCase (from APSM).

Semantics

This is a use case that includes the use case *Login*. Any registered user may attempt to activate their account before using the system. Activation includes a login attempt by default.

Associations

- *controller* : *ActivateAccountController*[1].

Constraints

- The association *controller* must be assigned to an instance of the class *ActivateAccountController* (from APSM).
- The association *include* must target at an instance of the class *LoginUseCase* (from APSM).

A.7 Admin (from APSM)

Generalizations

Member (from APSM).

Semantics

Admin is a special type of a User that has system-side access and represents the administrator of a system.

A.8 AdminComposite (from APSM)

Generalizations

MemberComposite (from APSM).

Semantics

The composite of the class Admin (from APSM).

A.9 Actor (from UML::UseCases)

Generalizations

BehaviourClassifier (from UML::Communications).

Semantics

Actors of different use cases are assigned different user interfaces because every actor may have a different viewpoint.

Associations

- *userInterface:UserInterface[0..1]* is the UI associated with this actor.

Presentation



Extensions

The association *userInterface:UserInterface[0..1]* is added.

A.10 Application (from Botterweck)

Generalizations

NamedElement.

Semantics

This represents a (web) application.

Associations

- *useCases : UseCase [1..*]* is the collection of use cases of the application, an application must have at least one use case.
- *startupUseCases : UseCase [1..*]* is the collection of use cases that contain the presentation units acting as the entry points of the application, i.e. a homepage; an application is supposed to have at least one startup use case.
- *siteMaps : SiteMap [1..*]* is the collection of site maps. An application must have one site map.
- *dataEntities : DataEntity [*]* is the collection of data entities.

- *dataComposites* : *DataComposite* [*] is the collection of data composites.
- *services* : *Service* [*] is the collection of services.

Presentation

At the visual level, an application is identified as a physical folder or directory that contains use case files.

Extensions

The Botterweck model is extended by adding the following associations:

- *useCases*
- *startupUseCases*
- *dataEntities*
- *dataComposites*
- *sitemaps*
- *services*

and by dropping the following associations:

- *dataModel* is replaced by direct associations with data entities.
- *webServices* is replaced by the association *services*. This is to ensure all types of services are accepted no matter if they have been defined as a web service or not.
- *stateMachines* is replaced by the association *useCases*. The reason is the APSM supports a use case-based application model.

A.11 CallEvent (from UML::Communications)

Generalizations

MessageEvent (from UML::Communications).

Semantics

Call events are used to call a controller operation from within the state machine that supports the behaviour of a use case. Call events are placed on transitions outgoing non-presentation states and are followed by choices that act based on the boolean return value of the controller operation. A call event is created per every operation defined at the PIM of the corresponding state machine.

Associations

- *transition:Transition[l]* is the transition that is the context of the call event.

Constraints

- Call events are only assigned to transitions outgoing from a non-presentations state. This is implemented in QVT relations.
- A call event may only call operations from a controller of a use case. This must be the use case that is the context of the state machine owning the transition to which the call event belongs. This is implemented in QVT relations.

Extensions

- The association, *transition* is added.

A.12 Cart (from APSM)

Generalizations

UIComposite (from Botterweck).

Semantics

Cart is the UI data structure to store session-related items. This is used for cases such as storing shopping items in an online shopping session or cases like books put in favourite list of an online library and course to be registered during an online registration session.

Associations

- *cartItems:CartItem[*]* is the collection of items that exist in the cart.

Presentation



A.13 CartItem (from APSM)

Generalizations

UIComponent (from Botterweck).

Semantics

A cart item is any UI component that could be added to a Cart such as shopping items, books or online courses.

Constraints

- The association end *uiComposite* must be of type *Cart*. This is implemented using QVT relations.

Presentation

CartItem is visualized based on the type of UIComponent chosen to realize it at the PIM level.

A.14 Class (from UML::Kernel)

Generalizations

Classifier (from Kernel).

Semantics

Classes may have attributes, operations or both. Classes are used to model data entities, data composites, services, controllers, UI components and UI composites.

A.15 Choice (from UML::BehaviourStateMachines)

Generalizations

Vertex (from UML::BehaviourStateMachine).

Semantics

A choice indicates a point of decision making based on a Boolean value returned from a controller operation.

Presentation

As in UML.

A.16 ContentOrientedUIElement (from Botterweck)

Generalizations

UIElement (from Botterweck).

Semantics

UI components that take a content such as text or image..

Associations

- *feedbacks:Feedback[*]* is the collection of other ContentOrientedUIElements as the feedback given to this element.
- *owner:UserComposite[1]* is the User, who created this content-oriented UI element.

Presentation

This element is displayed based on the actual UI component realizing it at the visual level.

Extensions

- The association *feedbacks* is added.
- The association *owner* is added.

A.17 ControlOrientedUIElement (from Botterweck)

Generalizations

UIElement (from Botterweck).

Semantics

This represents a UI element that causes a controlling action occurs such as a hyperlink, operation trigger or event.

Associations

- *navigation: Navigation[0..1]* is the navigation link that represents this element in a site map.

Extensions

- The association *navigation* is added.

A.18 ControllerClass (from APSM)

Generalizations

Class (from UML::Kernel).

Semantics

Controller is the logic centre of the use cases. Call events triggered by the transitions of state machines call operations from controller.

Associations

- *useCase:UseCase[1]* is the use case controlled by this controller.

A.19 DataAssociation (from APSM)

Generalizations

UIComponentAssociation (from APSM).

Semantics

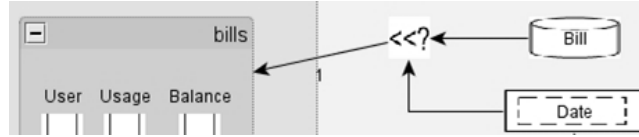
Data association identifies an association between UI components and data composites and infers a logical operation that affects one of the two sets or both. This is mapped to a controller operation to perform the required logic.

Associations

- *dataOperation:DataOperation[1]* is the operation handled by this association.
- *dataComposite:DataComposite[*]* is the collection of data composites involved in the operation.
- *uiComponent:UIComponent[*]* is the collection of UI components involved in the operation.

Presentation

A data association is displayed using the symbol representing the data operation as well as arrows from and to data composites and UI components indicating inputs to the operation as well as its effects on the data or UI. For example, in the following figure, a data association is used to populate the table *bills* using the data from the data composite *Bill* based the column *Date* filtered by the value of the variable *Date*.



A data association with two inputs and one affected UI component

A.20 DataComposite (from Botterweck)

Generalizations

Class (from UML::Kernel) and SecureItem (from APSM).

Semantics

A data composite is the transferable version of a data entity or a collection of data entities. Data composites are attribute-only classes that represent data from the database in the presentation and logic level.

Associations

- *composite:DataComposite[0..1]* is the data composite that is the base of the current data composite. This is used in cases, where a data composite such *Province* is the base of another data composite like *Town*.
- *singles:DataComposite[*]* is set to indicate the data composites combined with each other to form another data composite, such as different data used in the same table. This has one attribute:
 - *role:SingleRole[1]*, which defines the method the data composite is formed based on its constituents.
- *service:Service[1]* is the service affiliated with the low-level operations regarding the data composite; these are mainly the CRUD operations.
- *dataEntities:DataEntity[*]* is the collection of data entities this data composite represents.

Presentation



Extensions

- The generalization is altered from Botterweck::DataComponent to UML::Kernel::Class and APSM::SecureItem.
- All the associations from Botterweck are nulled.
- Associations *service*, *dataEntites*, *composite* and *singles* are added.
- DataComposite may implement the interface *SecureItem*.

A.21 DataEntity (from APSM)

Generalizations

Class (from UML::Kernel).

Semantics

Data entity is the low-level representation of data that is attribute-only and relates to at least one data composite. Data entities are considered equivalent to tables to objects at the programming level.

Associations

- *dataComposite:DataComposite[1]* is the data composite that carries the value of the data entity among higher levels.

A.22 DataOperation (from APSM)

Semantics

An association class between UIComponent and DataComposite. This defines the nature of data-dependency between a UI element and a data resource.

Attributes

- *logic:Logic* specifies the type of DataOperation and its order.

Presentation

Various depending on the type of operation as found in Table 4.

A.23 ErrorMessage (from APSM)

Generalizations

PageVariable (from UML::Kernel).

Semantics

Error message represents the content holder of regular error messages shown during a use case.

This is typically used in default presentation units displayed in response to a false return of a call event.

Presentation

This is displayed as other variables.

A.24 Event (from UML::Communications)

Generalizations

PackageableElement (from UML::Kernel).

Semantics

Events are used to handle UI triggers, link clicks and submit button.

Associations

- *transition:Transition[1]* is the transition that triggers the event.

Constraints

- Events can be only associated with transitions.

Extensions

- The trigger context is limited to transitions and hence a direct association with Transition is equipped.

A.25 Feedback (from APSM)

Generalizations

ContentOrientedUIElement (from Botterweck).

Semantics

Feedback is an interface that represents the different types of feedbacks one may provide to a Web 2.0 component such as rank, like, rate or comment. Feedback is considered a content-oriented UI element as well and hence can accept feedback.

A.26 FieldOperation (from APSM)

Semantics

An association class between two UIComponent classes. This defines the nature of dependency between two UI elements if any.

Attributes

- *logic:Logic* specifies the type of operation and its order.

Presentation

Various depending on the type of operation as found in Table 4.

A.27 FinalState (from UML::BehaviourStateMachine)

Generalizations

State (from UML::BehaviourStateMachine).

Semantics

Final state indicates a point, where the flow of a state machine may stop. This may represent transferring the flow to another state machine.

Associations

- *nextStep:StateMachine[0..1]* is the state machine, which the flow will go through after the current state machine.

Presentation

Final state is visualized as in UML - . UML note will be used to indicate a state machine associated with the final state as the next step.

Extensions

- The association *nextStep:StateMachine[0..1]* is added.

A.28 FoundItem (from APSM)

Semantics

FoundItem represents an individual content-oriented element that matches a search criteria.

Associations

- *search:Search[1]* is the search that resulted in this item.
- *hyperLink:Hyperlink[1]* is the hyperlink affiliated with the found item, which directs the flow to a presentation unit reflecting the contents of the found item.
- *theContent:ContentOrientedUIElement[1]* is the content-oriented Ui element affiliated with the search results.

Presentation

FoundItem can be shown as a hyperlink -  - or a formatted text -  -.

A.29 Hyperlink (from Botterweck)

Generalizations

ControlOrientedUIElement (from Botterweck).

Semantics

A hyperlink is handled as a control-oriented UI component; that is clicking on a hyperlink may cause a trigger calling a controller operation.

Presentation



Extensions

- Hyperlink is extended by being associated with FoundItem.

A.30 Image (from Botterweck)

Generalizations

ContentOrientedUIElement (from Botterweck).

Semantics

An image is considered a content-oriented UI element, which may contain a picture, present some information and get feedback.

Presentation



A.31 LockAccount (from APSM)

Generalizations

UseCase (from APSM::UseCases).

Semantics

LockAccount is a use case that extends the use case *Login*. Based on a defined number of unsuccessful login attempts in a specific period, this use case locks a specific user account.

Attributes

- *nAttempts:int* is the maximum number of login attempts allowed.
- *lastAttempt:Date* is the last unsuccessful attempt.
- *period:Date* is the period during which unsuccessful attempts are accumulated. The default value is the duration an inactive session remains valid.

Constraints

- The association *extend.extensionLocation.useCase* is set to an instance of type *LoginUseCase*.

A.32 Logic (from APSM)

Semantics

Whether an operation is a UI-level action or a service-level data query as well as the order of operation is specified here.

Attributes

- *queryType:QueryType* is this a data query and of what type?
- *actionType:ActionType* is this a non-data action and of what type?
- *order:Long* what is the order of this logic in presentation load process?

Presentation

Various depending on the type of operation as found in Table 4.

A.33 LoginAttempts (from APSM)

Generalizations

PageVariable (from APSM).

Semantics

LoginAttempts is a variable that holds the number of unsuccessful attempts in every session.

Associations

- *loginUseCase>LoginUseCase[1]* is the associated login use case.

A.34 LoginController (from APSM)

Generalizations

Controller (from APSM).

Semantics

This is the controller of the login use case.

Operations

- *login(username:String, password:String):boolean*, returns true if login using the given parameters *username* and *password* is successful, false otherwise.
- *hasAttemptsReached():boolean*, returns true if the number of unsuccessful login attempts is reached or false otherwise.

Constraints

- The association *useCase* must be assigned to an instance of the class *LoginUseCase* (from APSM).

A.35 *LoginUseCase (from APSM)*

Generalizations

UseCase (from APSM).

Semantics

This is the default login use case.

Constraints

- The association *controller* must be assigned to an instance of the class *LoginController* (from APSM).
- The association *include* must point to a target of an instance of the class *ViewHomePage* (from APSM).

A.36 *LogoutController (from APSM)*

Generalizations

Controller (from APSM).

Semantics

This is the controller of the default logout use case.

Operations

- *logout():boolean*, returns true if logout is successful and false otherwise.

Constraints

- The association *useCase* must be assigned to an instance of the class *LogoutUseCase* (from APSM).

A.37 *LogoutUseCase (from APSM)*

Generalizations

UseCase (from APSM).

Semantics

This is the default logout use case.

Constraints

- The association *controller* must be assigned to an instance of the class *LogoutController* (from APSM).

A.38 Member (from APSM)

Generalizations

User (from APSM).

Semantics

Member is a special type of a User that is registered.

A.39 MemberComposite (from APSM)

Generalizations

UserComposite (from APSM).

Semantics

The composite of the class Member (from APSM).

A.40 Navigation (from APSM)

Generalizations

NamedElement (from UML::Kernel).

Semantics

This represents a link between two different nodes in a site map.

Associations

- *source* : *NavigationNode*[1], the source of the navigation.
- *target* : *NavigationNode*[1], the target of the navigation.
- *control* : *ControlOrientedUIElement*[1], the control that causes the navigation to occur.

Extensions

- The association *source* is added.
- The association *target* is added.
- The association *control* is added.

A.41 NavigationNode (from APSM)

Generalizations

NamedElement (from UML::Kernel).

Semantics

This represents a node in a site map.

Associations

- *outgoings* : *Navigation[*]*, navigations outgoing this node.
- *incomings* : *Navigation[*]*, navigations ending at this node.
- *presentation* : *Presentation[1]*, the presentation represented by this node.

Extensions

- The association *outgoings* is added.
- The association *incomings* is added.
- The association *presentation* is added.

A.42 Operation (from UML::Kernel)

Generalizations

BehaviouredFeature (from UML::Kernel).

Semantics

This represents either a service operation or a controller operation.

Attributes

- *crudNature* : *QueryType[1]*, defines the type of operation.

Associations

- *callee* : *Operation[*]*, operations called by this operation.
- *caller* : *Operation[*]*, operations calling this operation.

Extensions

- The association *callee* is added.
- The association *caller* is added.

A.43 OperationTrigger (from Botterweck)

Generalizations

UIComposite (from Botterweck).

Semantics

For the most part, an operation trigger resembles a submit button. More generally, this is any trigger caused from the presentation that needs an action by the controller.

Associations

- *useCase* : *UseCase*[0..1], declares a use case that will be directly invoked by the operation trigger. This is used to model sequencing state machines.
- *controller* : *OperationTriggerController*[1], assigns an operation controller to this operation trigger.
- *presentation* : *Presentation*[1], the presentation holding this operation trigger.

Presentation

This is shown as a grouping of a number of UI components. The name of the group is considered as the name of the operation trigger. This group must be inside another group that represents a presentation.

Extensions

- The association *presentation* is added.
- The association *useCase* is added.
- The association *controller* is added.
- Generalization is changed from *UIElement* to *UIComposite*.

A.44 OperationTriggerController (from APSM)

Generalizations

ControllerOperation (from APSM).

Semantics

This is a controller operation that is directly triggered by the trigger event associated with an operation trigger.

Associations

- *trigger* : *Trigger[1]*, is the corresponding operation trigger.

Extensions

- The association *trigger* is added.

A.45 PageVariable (from APSM)

Generalizations

UIComponent (from Botterweck) and SecureItem (from APSM).

Semantics

A page variable is used to hold information pertaining a session, a page or the application.

A.46 Presentation (from Botterweck)

Generalizations

UIComposite (from Botterweck).

Semantics

A presentation is any unit of UI that represents a state of the application. This is realized based on design decisions pertinent to every specific platform. Options range from web pages to panels or user controls.

Associations

- *presentationState:State[0..1]* is the state in which the application interacts with the User using this presentation.
- *navigationNode:NavigationNode[0..1]* is the related node in the site map.
- *holder:UIComposite[0..1]* points to the UI composite that contains this presentation.

Extensions

- The association *presentationState* is added.
- The association *navigationNode* is added.
- The association *holder* is added.

A.47 QueryType (from APSM)

Semantics

This is an Enumeration that carries the default data query types that may be associated with UI components.

Attributes

Enumerations are as follows:

- none (=0), for exceptions when the type of query may not be identified.
- selectAll (=1), selects all objects/rows of a class/table.
- remove (=2), removes the specified object/row of a class/table.
- update (=3), updates the specified object/row of a class/table.
- selectSingle (=4), selects first object/row of a class/table matching the specified criteria.
- create (=5), creates one object/row of a class/table.
- updateField (=6), updates an attribute/column of an object/row of a class/table.

Presentation

At the visual level, each query type is identified using a specific symbol:

- selectAll <<
- remove -
- update >>
- selectOne ?
- create +
- updateField >>
- find <<?

Extensions

The Botterweck model is extended by adding the enumeration, QueryType.

A.48 Region (from UML::BehaviourStateMachines)

Generalizations

Namespace, RedefinableElement (from UML::Kernel).

Semantics

Region does not represent any specific meaning in the APSM.

A.49 RegisterController (from APSM)

Generalizations

Controller (from APSM).

Semantics

This is the controller of the register use case.

Operations

- *areEmailsEqual(email1:String, email2:String):boolean*, returns true if emails are equal.
- *arePasswordsEqual(password1:String, password2:String):boolean*, returns true if passwords are equal.
- *register():boolean*, returns true if registration is performed successfully.
- *email():boolean*, emails the activation link to the registering user.

Constraints

- The association *useCase* must be assigned to an instance of the class *RegisterUseCase* (from APSM).

A.50 RegisterUseCase (from APSM)

Generalizations

UseCase (from APSM).

Semantics

This is the default registration use case.

Constraints

- The association *controller* must be assigned to an instance of the class *RegisterController* (from APSM).

A.51 Search (from APSM)

Generalizations

Class (from APSM).

Semantics

This represents a single instance of a search.

Associations

- *tag:Tag[0..1]* is the tag that is associated with this search.
- *results:FoundItem[0..n]* is the collection of items resulted from this search.

Extensions

This class is added.

A.52 SecureItem (from APSM)

Generalizations

Interface (from UML::Interfaces).

Semantics

A secure item is any piece of logic, data or presentation that is securely accessible. Items should implement this interface to become secure.

Operations

- *secure() : String* secures the item.
- *unsecure():String* removes the security mechanisms from this item.

Extensions

This interface is new.

A.53 Service (from APSM)

Generalizations

Class (from APSM) and SecureItem (from APSM).

Semantics

A service introduces access to operations that facilitate working with external resources such as database, other applications or devices.

Associations

- *dataComposite:DataComposite[0..1]* is the data composite this service manages.

Extensions

This class is new.

A.54 SignalEvent (from UML::Communications)

Generalizations

Behaviour (from APSM).

Semantics

A signal event carries a message, which could be a container for some information or a holder of event parameters passed along a transition.

Associations

- *transition:Transition[1]* is the transition, this signal event is passed along.

Extensions

- The generalization is changed from MessageEvent to Behaviour.
- The Association *transition* is added.

A.55 SingleRole (from APSM)

Generalizations

Enumeration (from UML::Kernel).

Semantics

This enumeration contains the literals to define the relationship of two data composites that are used together to fill out the contents of a UI component. A data composite can be the base of

selecting data from another data composite such as a province being the base of selecting the list of towns.

Attributes

- *selectionBase* (=0), the data composite is the base of selection, e.g. a province.
- *selectionSource* (=1), the data composite is the source of data, e.g. list of towns.

Extensions

This enumeration is new.

A.56 SiteMap (from APSM)

Generalizations

Class (from UML::Kernel).

Semantics

This is a map of the web site that acts as the front-end of the web application.

Associations

- *userInterface:UserInterface[1]* is the user interface this map represents.
- *navigationNodes:NavigationNode[0..n]* is the collection of nodes in this map.

Extensions

This class is new.

A.57 State (from UML::BehaviourStateMachines)

Generalizations

Vertex (from UML::BehaviourStateMachines).

Semantics

A state in a web application; this is either a presentation or a logic state.

Attributes

- *loginRequired:boolean* indicates whether accessing this state requires a login or not.

Associations

- *presentation:Presentation[0..1]* is the presentation assigned to this state if any.

Presentation



Extensions

- The attribute *loginRequired* is added.
- The association *presentation* is added.

A.58 StateMachine (from UML::BehaviourStateMachines)

Generalizations

Behavior (from UML::BasicBehaviours).

Semantics

A state machine represents the behaviour of a use case.

Associations

- *useCase:UseCase[0..1]* is the use case this state machine models the behaviour of, if any.

Presentation

State machine is presented as a file containing the visual presentation of UML state machines and presentation models.

Extensions

The association *useCase* is added.

A.59 StaticText (from Botterweck)

Generalizations

ContentOrientedUIElement (from Botterweck).

Semantics

This represents a textual element that appears on a presentation and may not be edited.

Presentation



A.60 StructuredText (from Botterweck)

Generalizations

ContentOrientedUIElement (from Botterweck).

Semantics

This represents a textual element composed of formatting elements that appear on a presentation.

Presentation



A.61 Tag (from APSM)

Generalizations

NamedElement (from APSM).

Semantics

A textual label a user assigns to a content-oriented element to categorise the element with other elements sharing the same tag.

Associations

- *owner:UserComposite[0..1]* who created the tag.
- *taggedContents:ContentOrientedUIElement [0..n]* the elements sharing this tag.
- *taggedSearch:Search[0..1]* is the search phrase triggered by this tag.

A.62 Transition (from UML::BehaviourStateMachines)

Generalizations

Namespace (from UML::Kernel).

Semantics

A transition indicates a state change in the web application, which is either a navigation path between two different presentations or a control switch to/from a state that performs some logical operations.

Presentation

Transition is represented by a unidirected arrow from the source state to a target state as appears in UML.

A.63 UIComponent (from Botterweck)

Generalizations

NamedElement (from UML::Kernel).

Semantics

A UIComponent represents any object that is a part of a presentation and shares either a piece of information or event with the User.

Associations

- *holder:UIComposite[1]* is the composite holding this component.

Presentation

A UI component is presented based on its concrete type.

Extensions

- Association *holder* is added.

A.64 UIComponentAssociation (from APSM)

Generalizations

Class (from UML::Kernel).

Semantics

A UIComponentAssociation represents an operation relationship between two UI and/or data composites.

Associations

- *source:UIComponent[1]* is the supplier of the operation.
- *target:UIComponent[1]* is the producer of the operation.
- *dataOperation:DataOperation[1]* is the operation.

Presentation

This is shown using one of the operation symbols.

A.65 *UIComposite (from Botterweck)*

Generalizations

UIComponent (from Botterweck).

Semantics

A UIComposite represents a UI Component, which can contain other UI Components. Examples are forms, tables and pages.

Associations

- *uiComponents:UIComponent[0..n]* is the collection of contained UI components.

Presentation

UI composites are represented as a grouping of the contained UI Components..

Extensions

Association *uiComponents* added.

A.66 *UnlockAccount (from APSM)*

Generalizations

UseCase (from APSM::UseCases).

Semantics

UnlockAccount is a use case that unlocks a specific user account.

A.67 *UseCase (from APSM)*

Generalizations

BehaviourClassifier (from UML::Communications).

Semantics

Use case is visual description of a piece of functionality of the web application. It normally contains a start state, one or more presentation states, transitions between states and final states representing other related use cases. The description, also, contains the contents of presentation states.

Associations

- *stateMachine:StateMachine[1]* is the state machine describing the behavior of this use case.
- *controller:ControllerClass[1]* is the controller of this use case.

Extensions

- The association *stateMachine* includes the association *ownedBehavior*.
- The association *controller* is added.

A.68 User (from APSM)

Generalizations

NamedElement (from UML::Kernel).

Semantics

User represents any user of the web application.

A.69 UserInterface (from APSM)

Generalizations

UIComposite (from Botterweck).

Semantics

A User Interface is a set of presentations, components and contents that is subject to interact with the User. Every application may have a few user interfaces each created per an actor.

A.70 UserName (from APSM)

Generalizations

PageVariable (from APSM).

Semantics

This is a page variable used to store the current username at the client side.

A.71 Validator (from APSM)

Generalizations

None (from APSM).

Semantics

Validator is an interface implemented by *ContentOrientedUIElements* to validate their contents.

Operations

validate(ValidationRule[]):String accepts a set of validation rules and applies them to the content-oriented UI element. The returning value is defined as String to allow clients using it for the desired purpose.

Extensions

The interface *Validator* is added.

A.72 ValidationRule (from APSM)

Generalizations

NamedElement (from UML::Kernel).

Semantics

ValidationRule represents an abstract rule used for validating the contents of a ContentOrientedUIElement.

Extensions

The class *ValidationRule* is added.

A.73 Vertex (from APSM)

Generalizations

NamedElement (from UML::Kernel).

Semantics

A vertex is either a state or choice in an APSM state machine.

A.74 Video (from Botterweck)

Generalizations

ContentOrientedUIElement (from Botterweck).

Semantics

This is the container or a division of a web presentation that holds a video.

A.75 ViewHomepageController (from APSM)

Generalizations

Controller (from APSM).

Semantics

This is the controller of the default logout use case.

Operations

- *logout():boolean*, returns true if logout is successful and false otherwise.

Constraints

- The association *useCase* must be assigned to an instance of the class *ViewHomepageUseCase* (from APSM).

A.76 ViewHomepageUseCase (from APSM)

Generalizations

UseCase (from APSM).

Semantics

This is the default use case to view the homepage of a logged-in user.

Constraints

- The association *controller* must be assigned to an instance of the class *ViewHomepageController* (from APSM).

Appendix B. **DILMAJ CASE STUDY**

The case study is a Web 2.0 application [Dilmaj2010] that is a free, web-based, collaborative, multilingual project about languages and language development supported by a non-profit group of developers. Many domestic languages are challenged in modern times. Dilmaj is a central place that users contribute to the development of challenged languages, and their relation to other languages through online and open collaboration by virtually all people who speak the language. Users may join, suggest terms, comment on others' suggestions or rank the entries. Figure 84 shows a use case diagram that includes core use cases of Dilmaj. The diagram contains main use cases that define the pivotal functionality of the application. Three main actors are the User, Member, i.e. a registered User, and Admin.



Figure 84 The Use Case Diagram of Dilmaj

In this chapter, use cases of Dilmaj are defined using the input language of MODEWIS. Every use case is also presented at the APSM level. The *apsm* of each use case is presented in two parts:

1. The *apsm* state machine: this state machine is the copy of the *pim* state machine plus the automatically added states and events. States are mainly added for the following reasons:
 - In order to handle a required behaviour by calling a controller operation. Such states are named “run state” concatenated with a number that indicates the order of the operation.

- In order to show an error message. Every choice is equipped with two outgoing transitions guarded with either “true” or “false”. The “false” guarded outgoing transition ends at an error state named “error state” concatenated with a number. Error states have two outgoing transitions: one that carries a “cancel” signal event, which normally leads to a final state, and another carrying a “retry” signal event, which returns the flow to the rune state preceding the choice that resulted in a false transition.
2. The *apasm* classes that are composed of the following:
- The controller of the state machine including all the operations called from run states. These operations are dependent to service classes.
 - Service classes, which call data services required to manage the data flow of the state machine. Services are dependent to data composites.
 - Data composites, which are used as parameters of controller and service operations to carry the contents between different layers. For data conversion purposes, data composites are dependent to data entities.
 - Data entities. Data entities hold the attributes inferred from that specific *apasm*. The same data entity may have different attributes in different *apsms*. Once integrated, attributes and operations of all classes are uniquely collected under their corresponding class.

B.1 Use Case: View Dilmaj

This is the startup use case of Dilmaj. Figure 85 shows the UI model of this use case. As the figure shows, all actors have access to this use case. The state machine is composed of one presentation state *View Dilmaj*. Three outgoing transitions end at final states annotated with the names of three use cases, *View Homepage*, *View Comments* and *View Use Case*. A User might also rank an entry and remain in the same presentation. The main presentation *View Dilmaj* has three other presentations within it. The main part is the table *Terms*, which is composed of several columns. As the model suggests, this table is populated by all instances of the data composite *Term*.

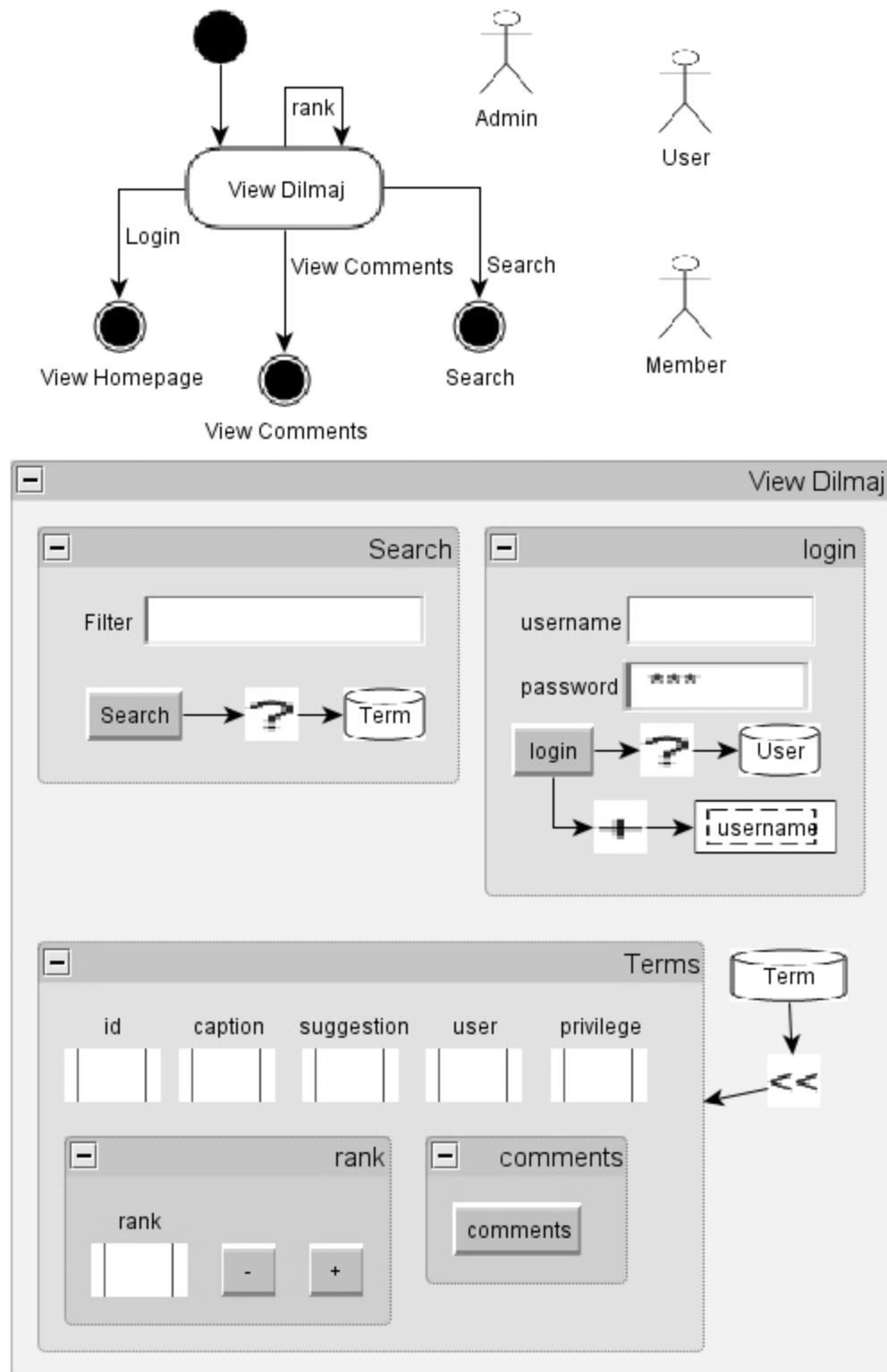


Figure 85 Use Case, View Dilmaj

Figure 86 shows the *apsm* model of Figure 85. The state machine in this figure, have several new elements. The state, *Run State 1* directs a call to the controller operation, *populateTermsTable*. This call event happens on the transition outgoing from the state, *Run State 1*. That transition leads to a choice. If the operation is successful, the navigation leads to the state *View Dilmaj*. But in case of any errors, the navigation leads to the state *Error State 1*, from which the User may choose to retry or cancel. These are represented as signal events. The state, *View Dilmaj* may lead to a number of (final) states. These states have other use cases attached to them.

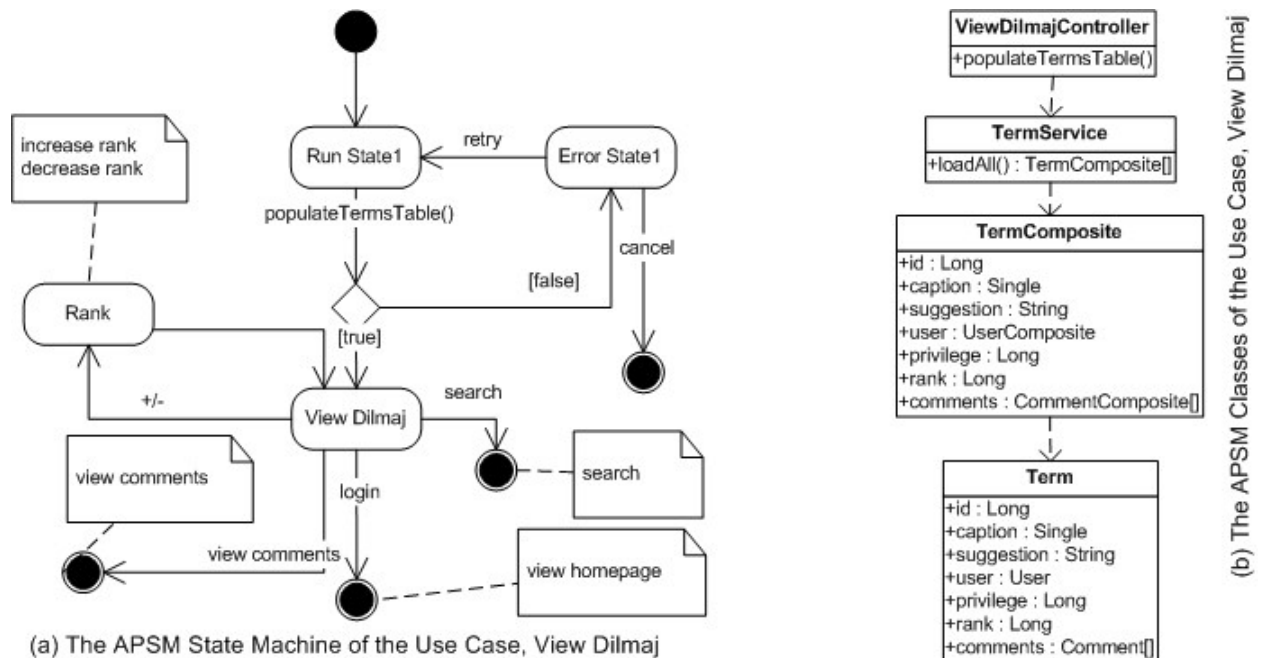


Figure 86 The *apsm* of the Use Case, *View Dilmaj*

Figure 86 also shows that a controller, *ViewDilmajController* is created. This controller has dependency to the service *TermService*. From the model of Figure 85, only one operation is discovered and added to this service that is the operation, *loadAll*. Operation *loadAll* loads all instances of the entity *Term*. To this end, *TermService* has dependency to the *TermComposite*, which in turn has dependency to the class *Term*. The attributes of the class *TermComposite* are defined based on the UI components found within the table, *Terms* in Figure 85.

B.2 Use Case Login

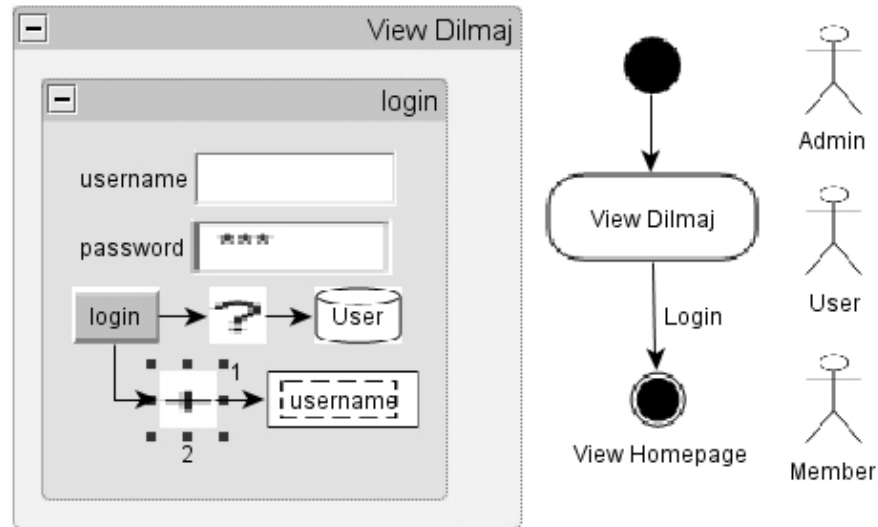


Figure 87 Use Case, Login

Figure 87 shows the input model of the use case *Login*. The button *login* uses the information within *username* and *password* inputs to first query the data composite *User*. Then the value of the input *username* is added to the variable *username*. As the state machine suggests the operation trigger *Login* leads to the page *View Homepage*. Figure 88 shows the *apsm* version. It has the states, *run state1*, *error state1*, a choice and their corresponding transitions automatically added to the state machines.

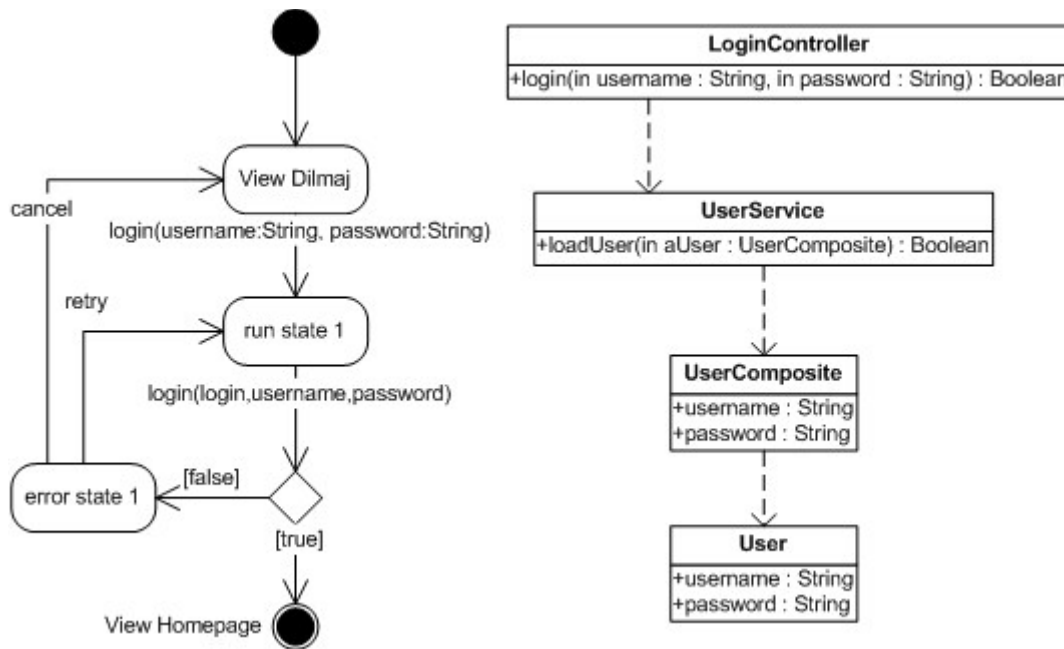


Figure 88 The *apsm* of the Use Case, *Login*

B.3 Use Case View Homepage

As shown in Figure 89, this use case browses a member's homepage once the login is successful. The presentation *View Homepage* has two tables. The table *Terms* is a list of all terms but the table *My Terms* contains a list of terms created by the member. As the figure shows, in the former case, a *loadAll* operation is used, while in the latter case the operation is *loadSome*. Operation *loadSome* needs a filter, which is, in this case, the value of the variable *username*. Figure 90 shows the *apsm* version of the use case *Login*. An interesting aspect of this state machine is the appearance of the variable *username* on its first transition. This variable is used to filter the list of loaded terms to those created by the logged-in User.

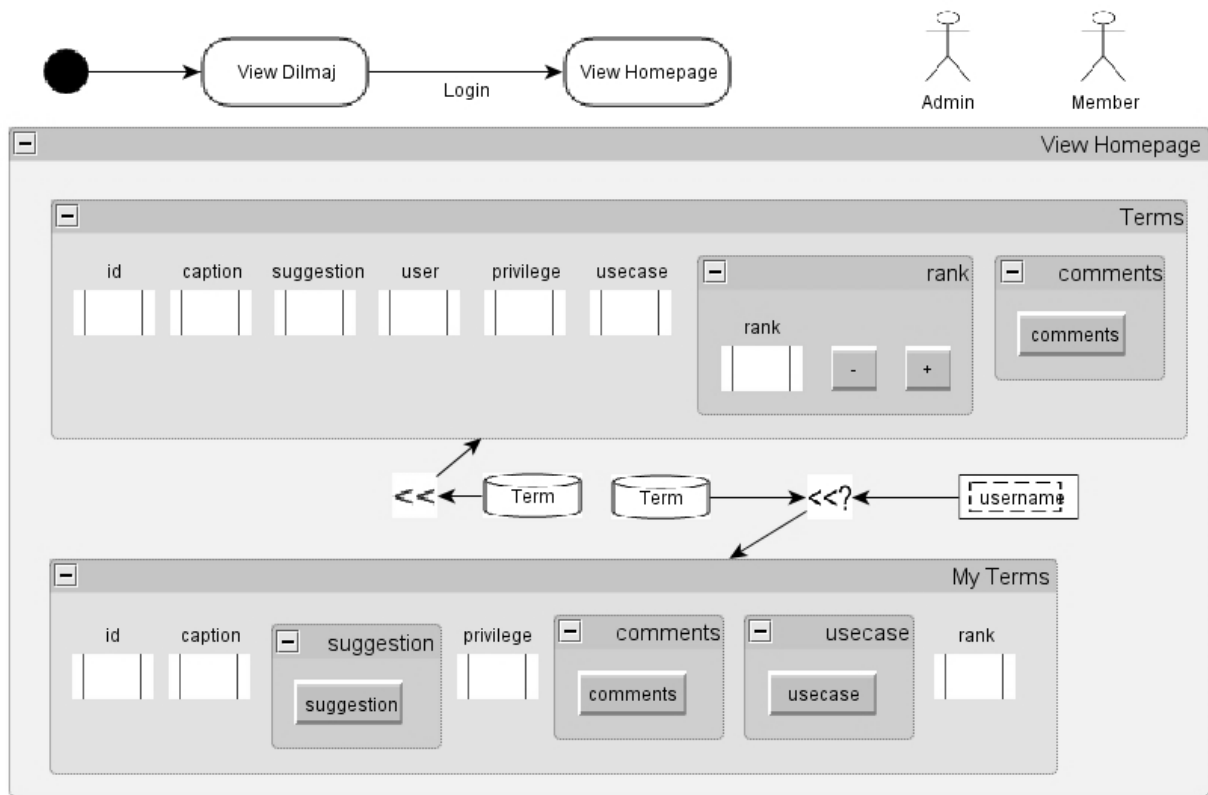


Figure 89 Use Case, View Homepage

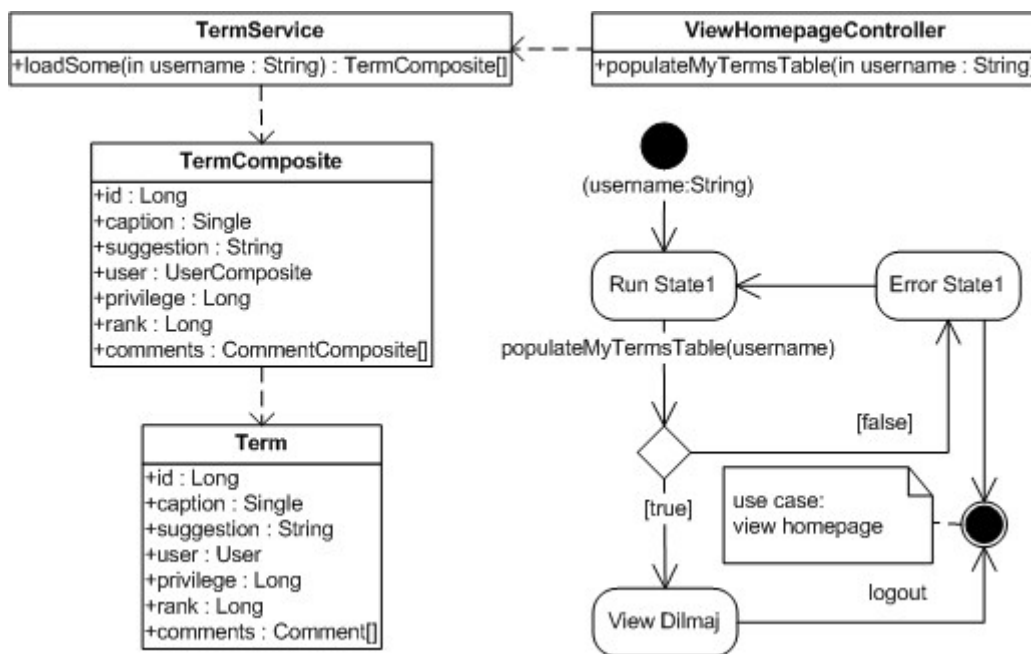


Figure 90 The *apsm* of the Use Case, View Homepage

B.4 Use case Logout

As Figure 91 shows, this use case simply resets the value of the variable *username* and returns to the state machine *View Dilmaj*. The operation trigger *logout* is a part of the presentation *View Homepage*. This was not shown in Figure 88. It is important to note that while it is possible to include all components of a composite in one model, this is not necessary because the final model is an integration of all presentations. Having only components related to the logic of each use case within its UI model simplifies models.

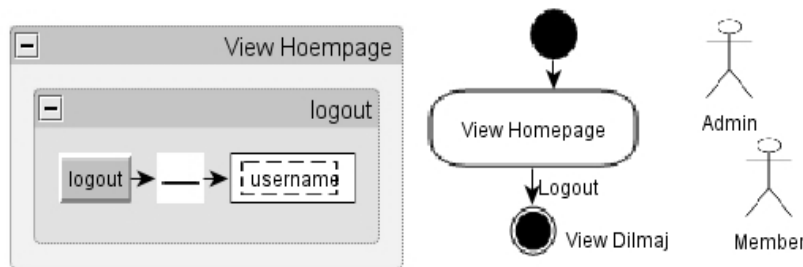


Figure 91 Use Case, Logout

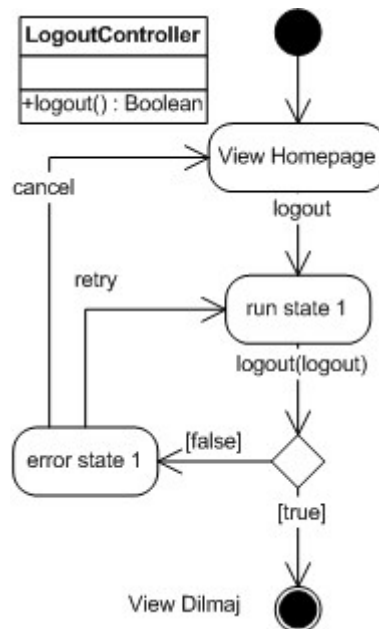


Figure 92 The *apsm* of Use Case, Logout

B.5 Use Case Add Comment

This use case is selected by a User in order to add comments, while being in the main page. The presentation *View Comments* is loaded with comments of the selected term. This is based on the variable *selectedTerm* that is set once the operation trigger, *comments* is fired in presentation *View Dilmaj*. Figure 93 shows the input model and Figure 94 is the *apsm* model.

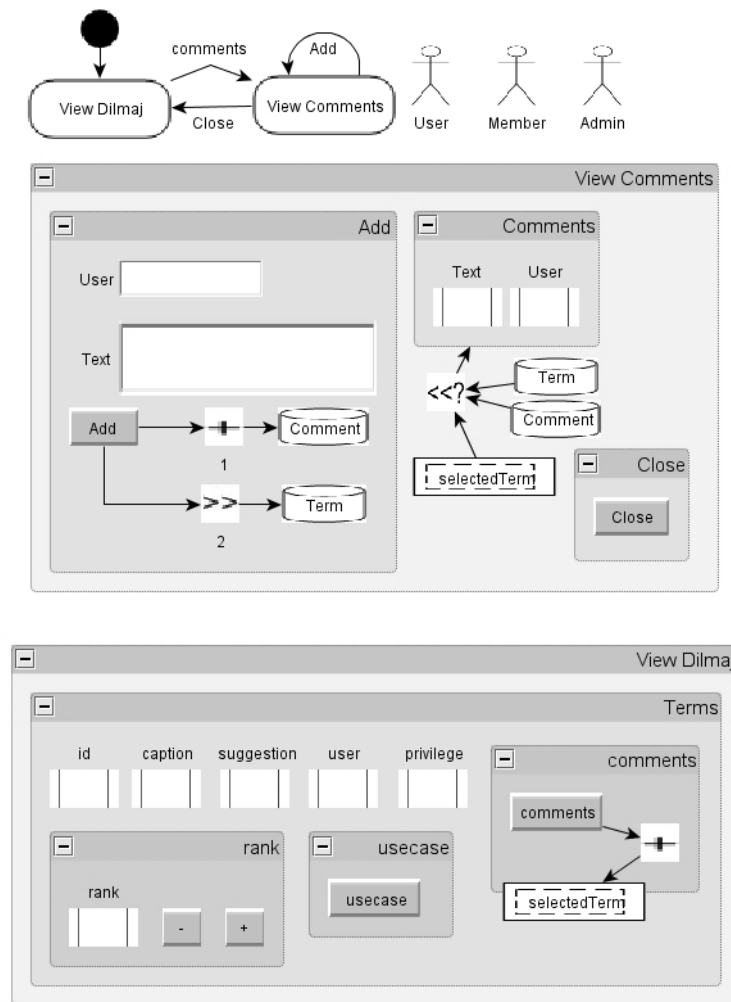
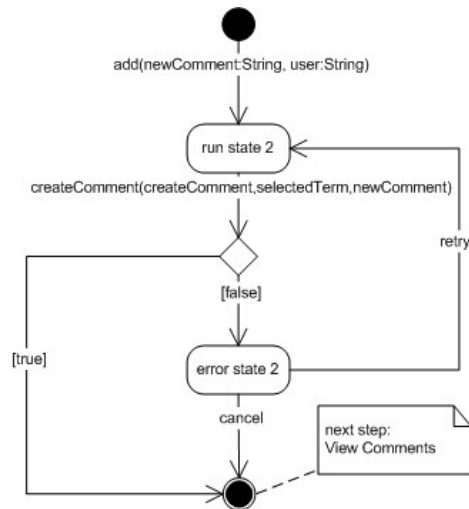
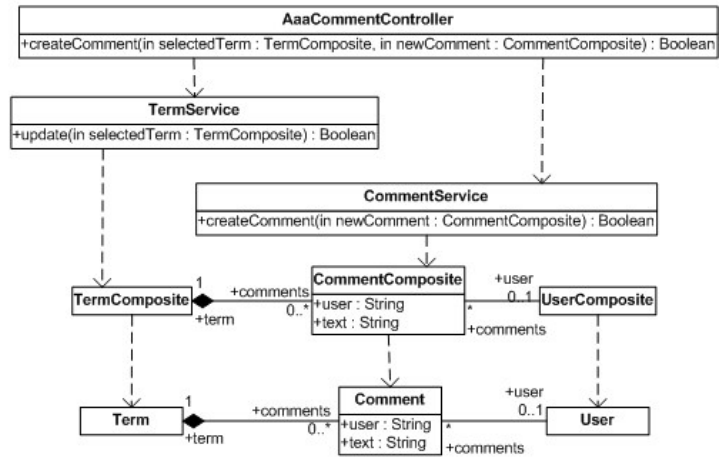


Figure 93 Add Comment



(a) The APSM State Machine of the Use Case, Add Comment



(b) The APSM Classes of the Use Case, Add Comment

Figure 94 The *apsm* of the Use Case, *Add Comment*

B.6 Use Case View Comments

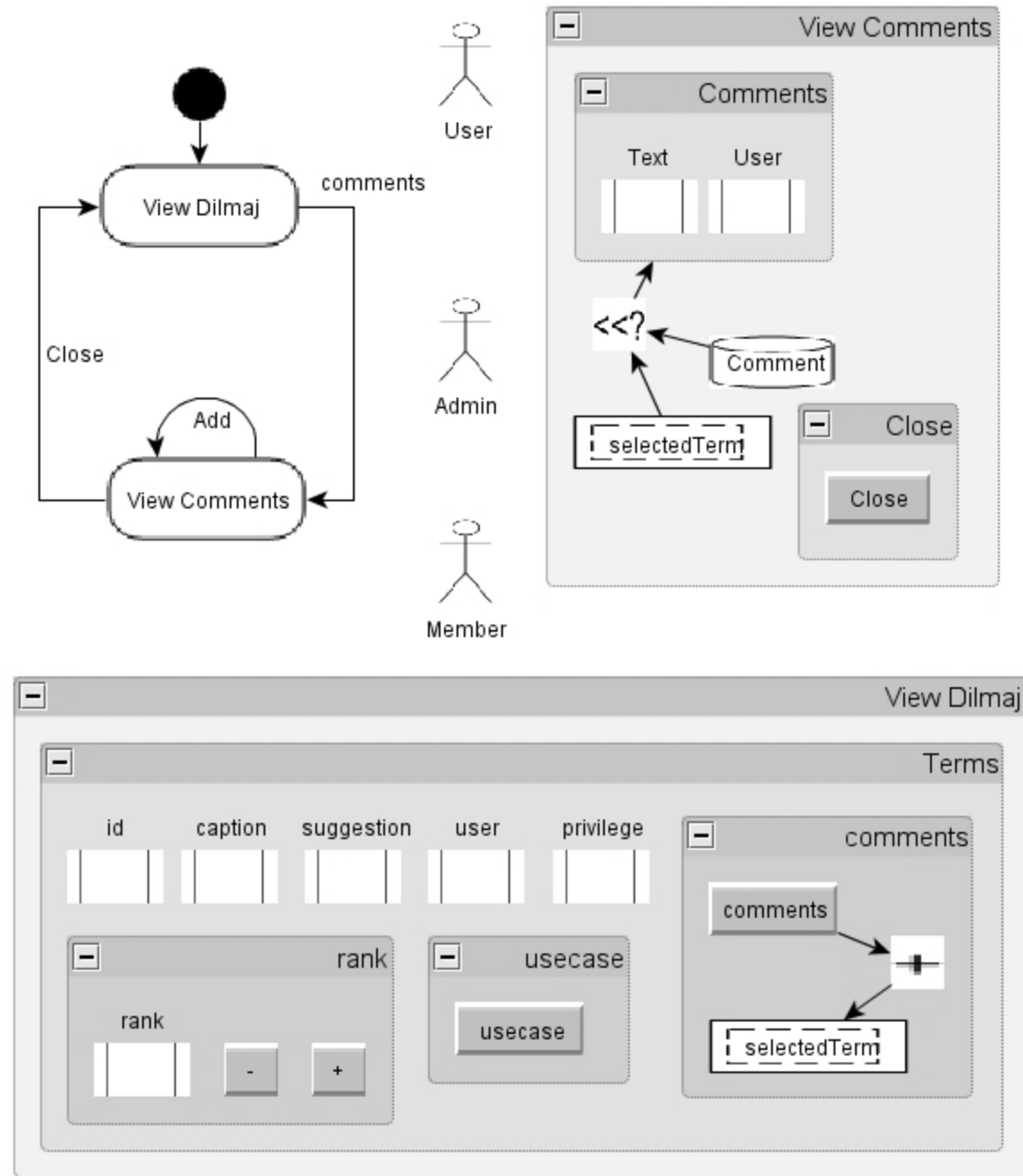


Figure 95 View Comments

This use case intends to show the comments associated to an entry. As Figure 95 shows, the operation trigger *Comments* stores the selected row of the table *Terms* into the variable *selectedTerm*. This variable is then used by the presentation, *View Comments* to load all comments of the selected term.

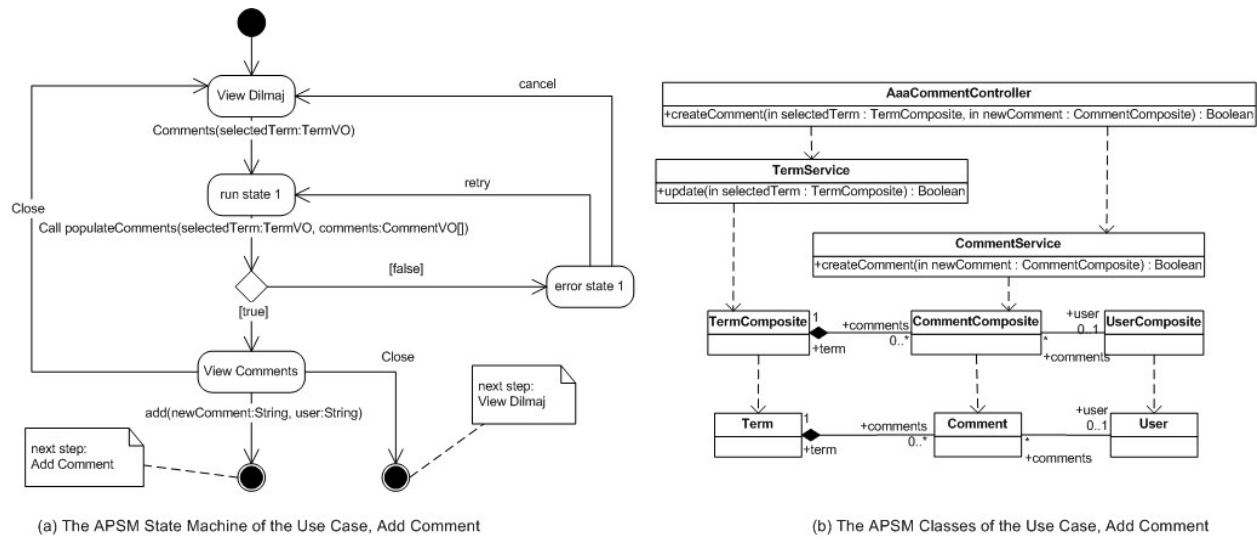


Figure 96 The *apsm* of the Use Case, *View Comments*

B.7 Search Use Case

This use case is started from the presentation *View Dilmaj*. The button *Search* runs a search query over the data composite *Term*. The results from the search operation are then stored in the variable *foundTerms*. This variable is then used to populate the table *Terms* in the presentation *Search*.

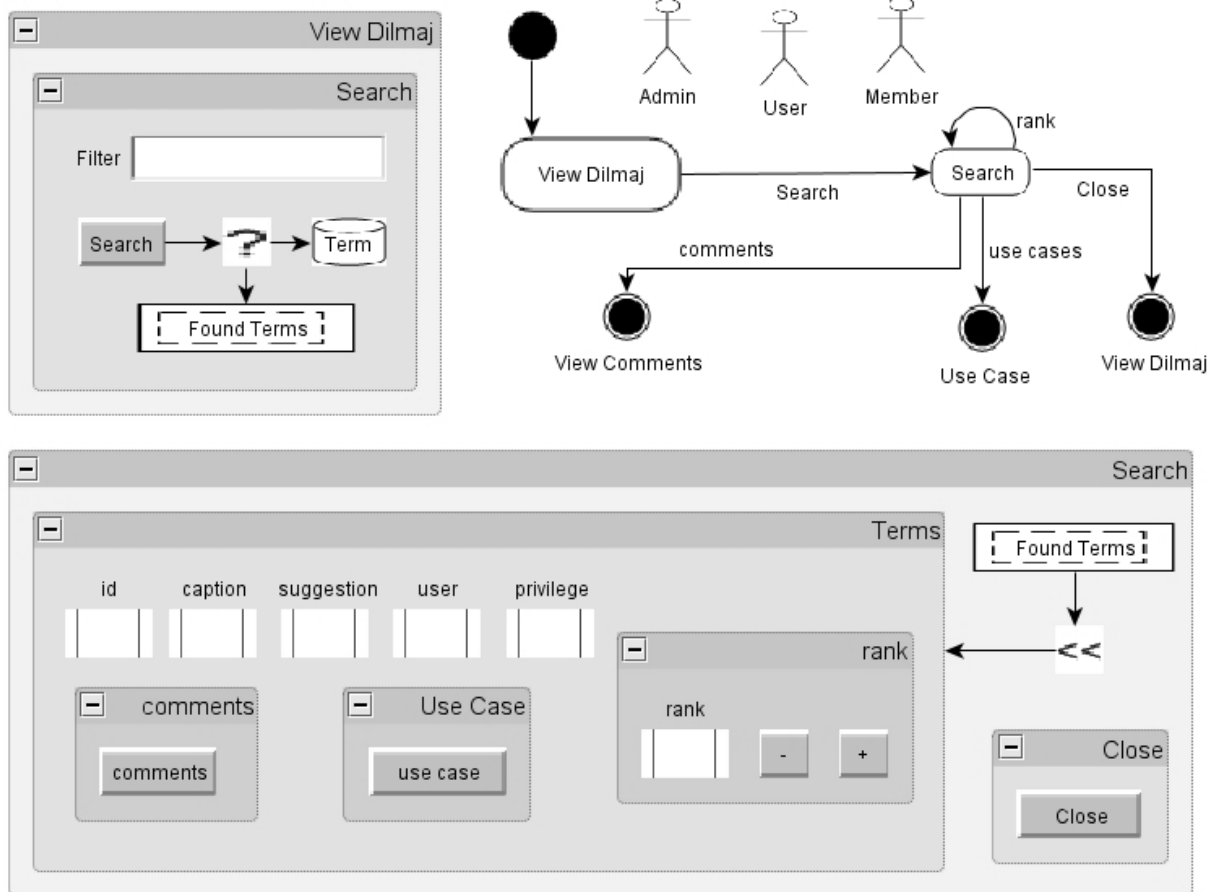
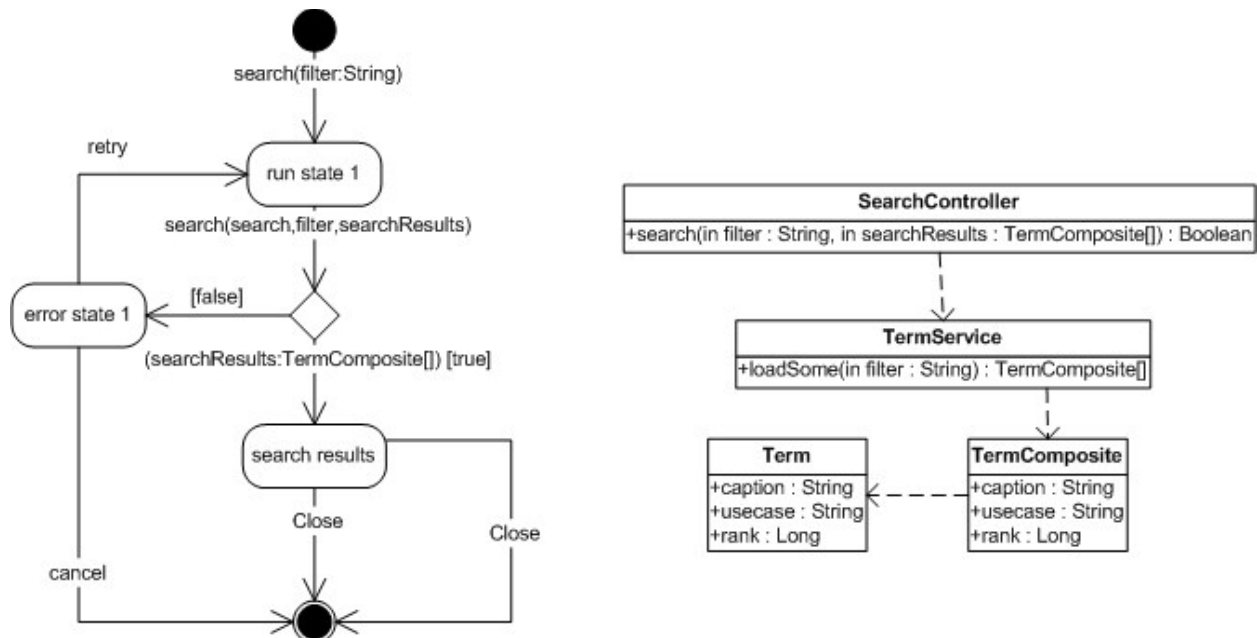


Figure 97 Search



(a) The APSM State Machine of the Use Case, Search (b) The APSM Classes of the Use Case, Search

Figure 98 The *apsm* of the Use Case, Search

B.8 Ranking

This use case describes the tasks of increasing and/or decreasing the rank of an entry (Figure 99).

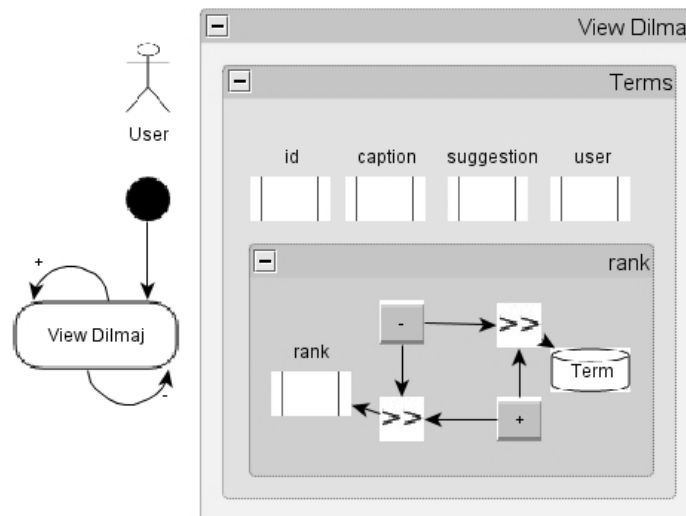


Figure 99 Ranking Use Cases

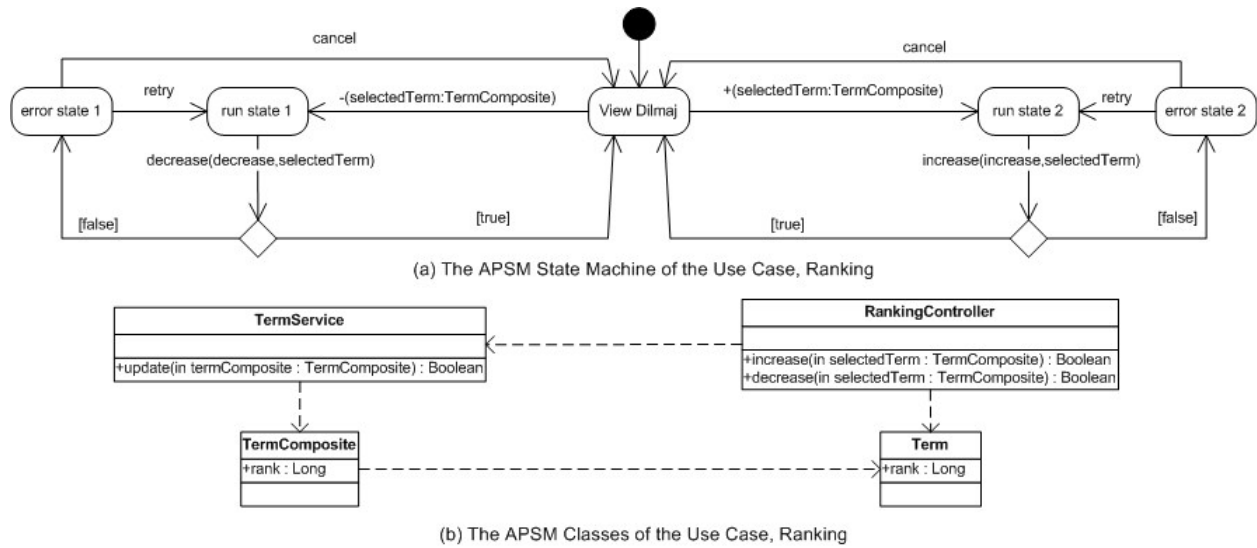


Figure 100 Ranking Use Cases

B.9 Privilege

A privilege is a special kind of ranking. A privilege is set when a registered user ranks an entry. A member cannot assign more than one privilege to an entry while non-registered users may rank repeatedly.

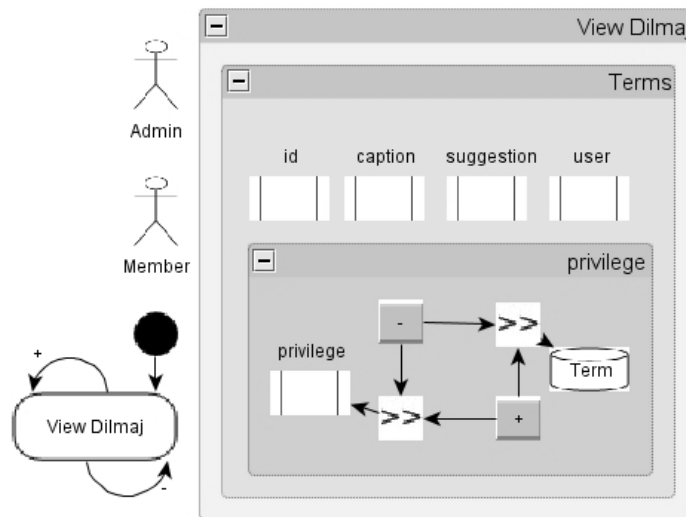
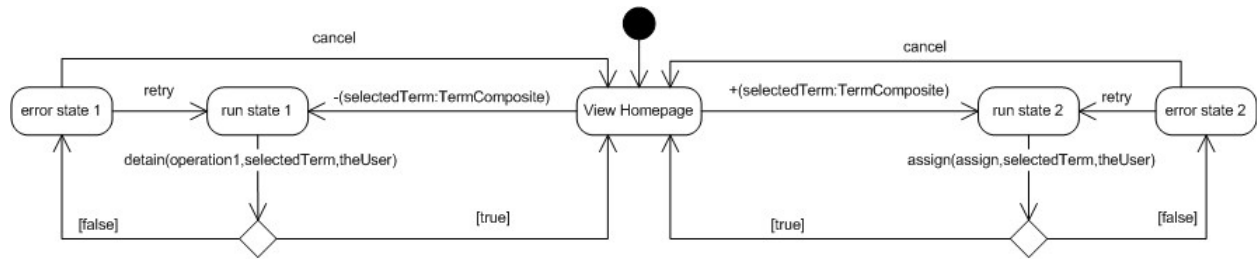
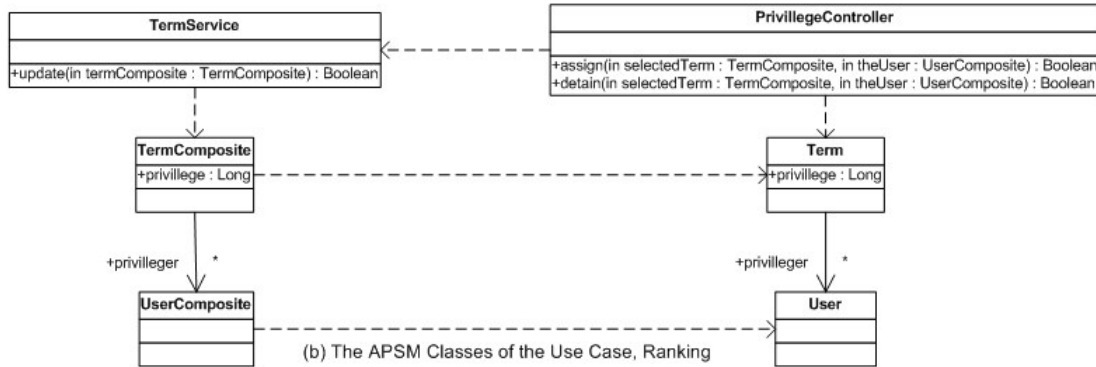


Figure 101 Set/Reset Privilege



(a) The APSM State Machine of the Use Case, Ranking



(b) The APSM Classes of the Use Case, Ranking

Figure 102 The *apsm* of the Use Cases, *Set/Reset Privilege*

B.10 Edit Details

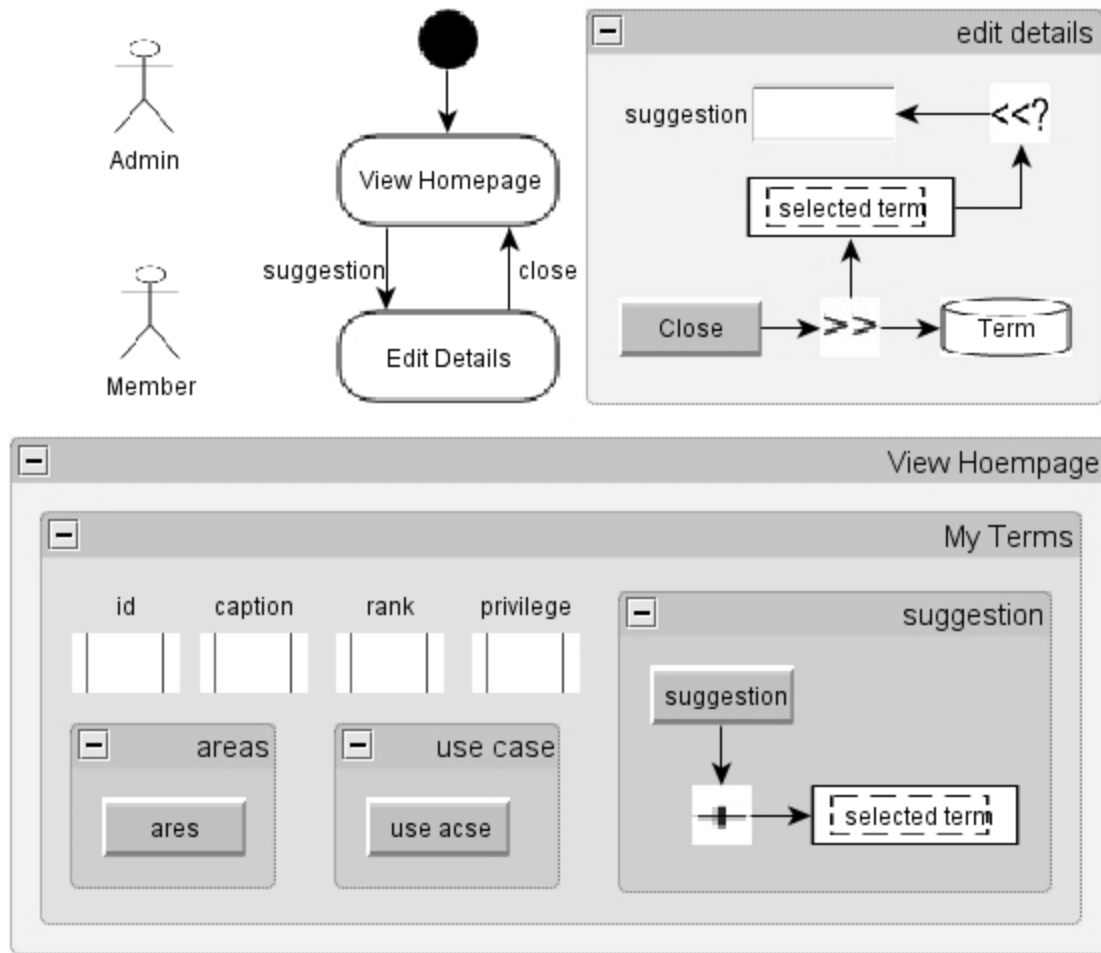


Figure 103 Edit Details

The use case *Edit Details* includes two steps. The selected term is stored within the variable, *selected term*. Then, the suggestion of the selected term is shown such that it can be edited.

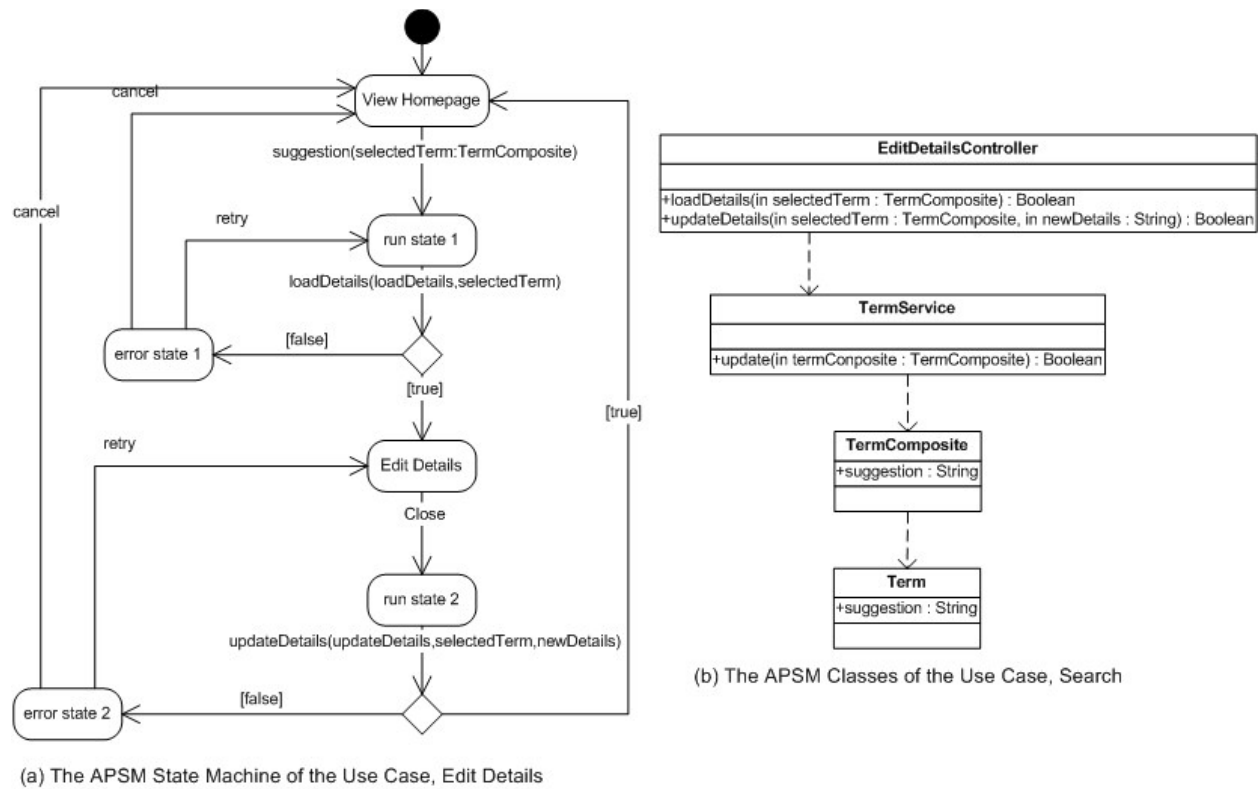


Figure 104 The *apsm* of the Use Case, *Edit Details*

B.11 Add Area

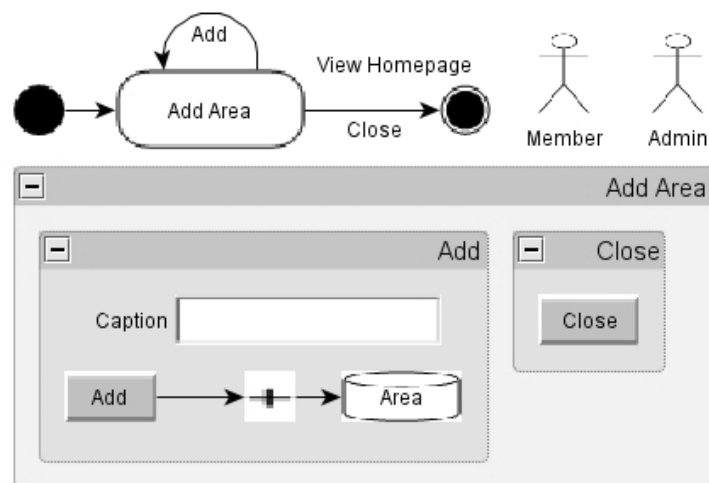
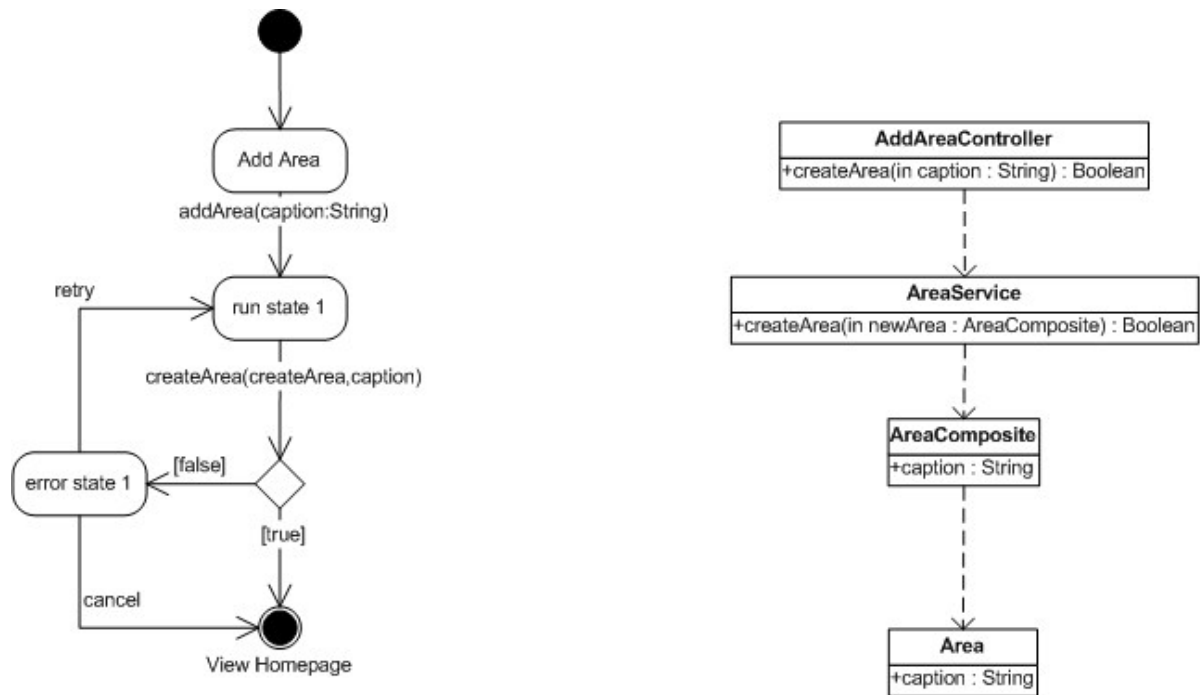


Figure 105 Add New Area

This use case adds an area to the list of available areas.



(a) The APSM State Machine of the Use Case, Add Area (b) The APSM Classes of the Use Case, Add Area

Figure 106 The *apsm* of the Use Case, *Add Area*

B.12 Edit Areas

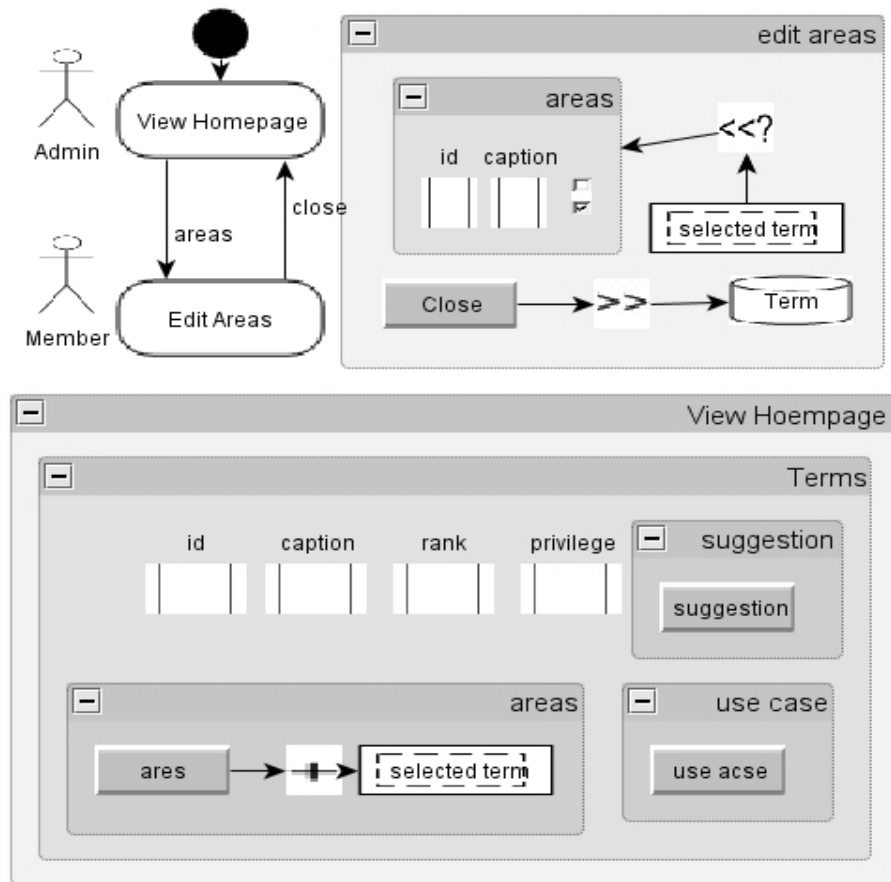
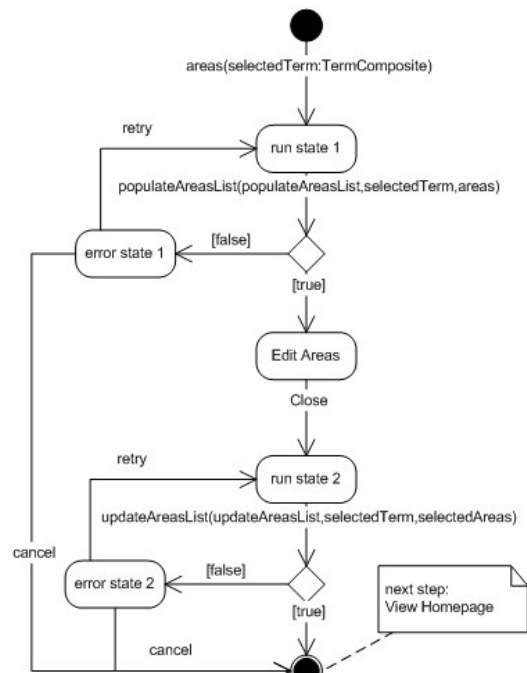
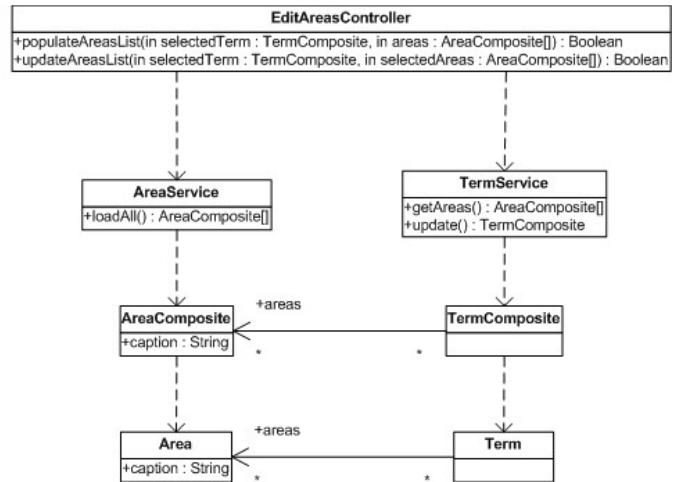


Figure 107 Edit Areas

This use case enables editing the areas assigned to a term.



(a) The APSM State Machine of the Use Case, Edit Areas



(b) The APSM Classes of the Use Case, Edit Areas

Figure 108 The *apsm* of the Use Case, *Edit Areas*

B.13 Edit Use Case

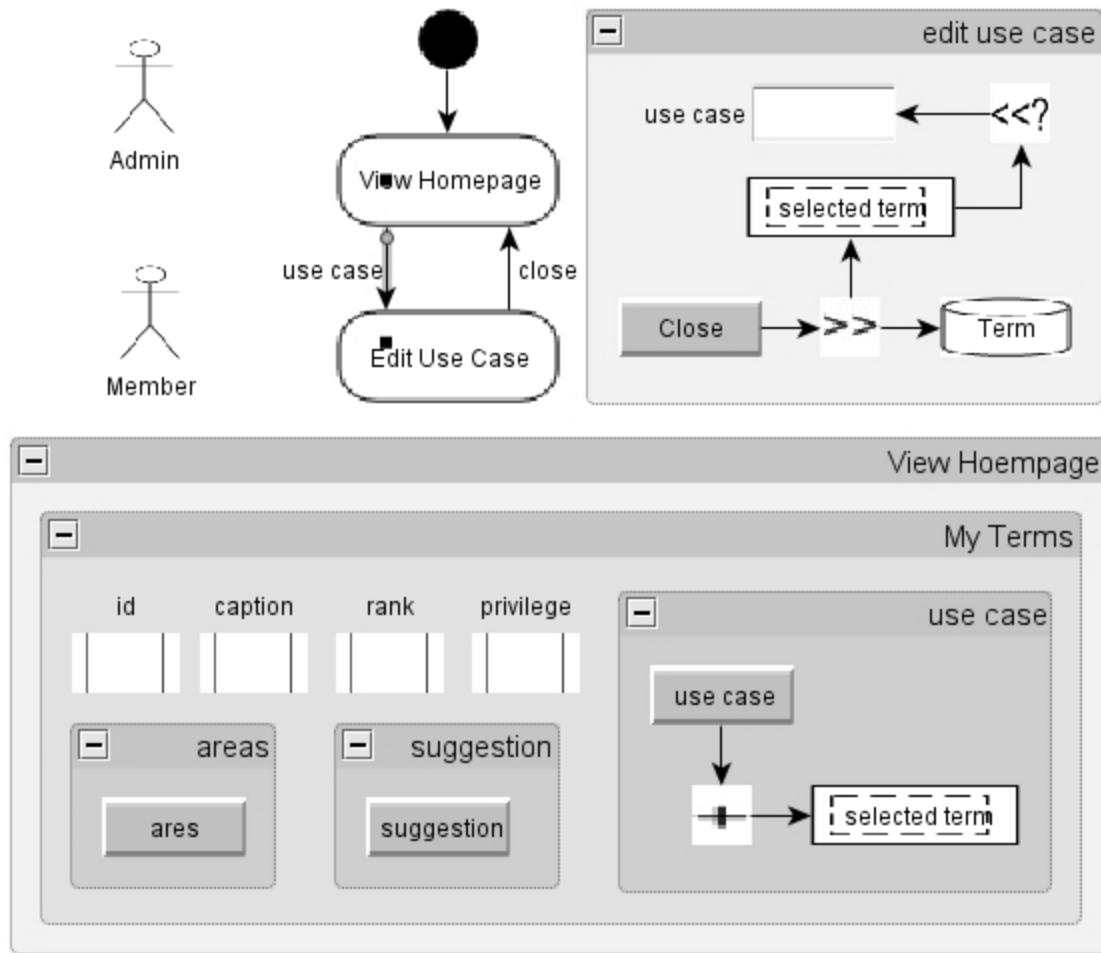


Figure 109 Edit Use Case

This use case edits the use case defined for every term.

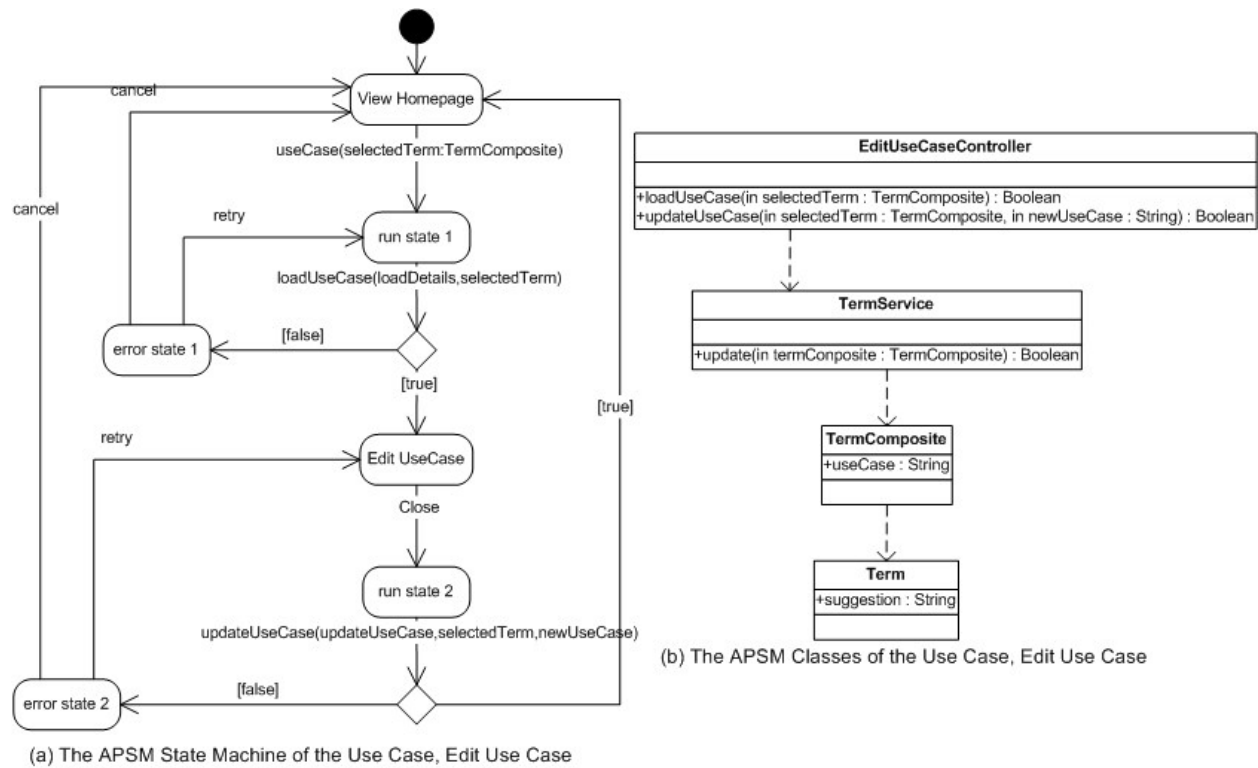


Figure 110 The *apsm* of the Use Case, *Edit Use Case*

B.14 Register

The User can register through this use case. Figure 111 shows the input model, where the User enters a username, password and email. As the figure shows, input fields, *email* and *password* are repeated and the value of each field must match with the one of the same name. Figure 111 also shows that an email must be sent to the User. The content of this email could be a confirmation message or an activation link.

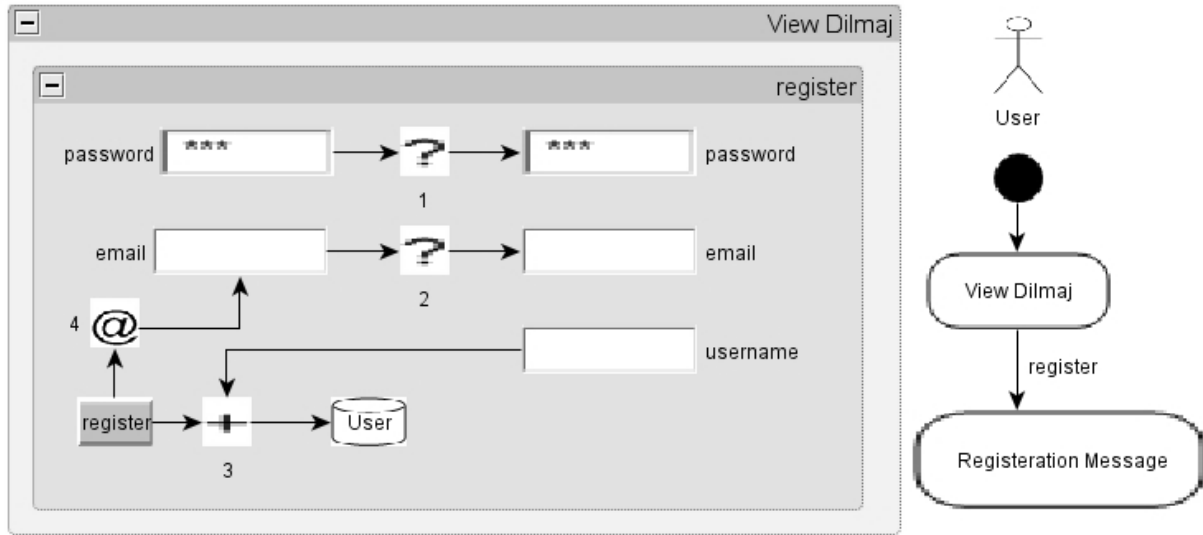
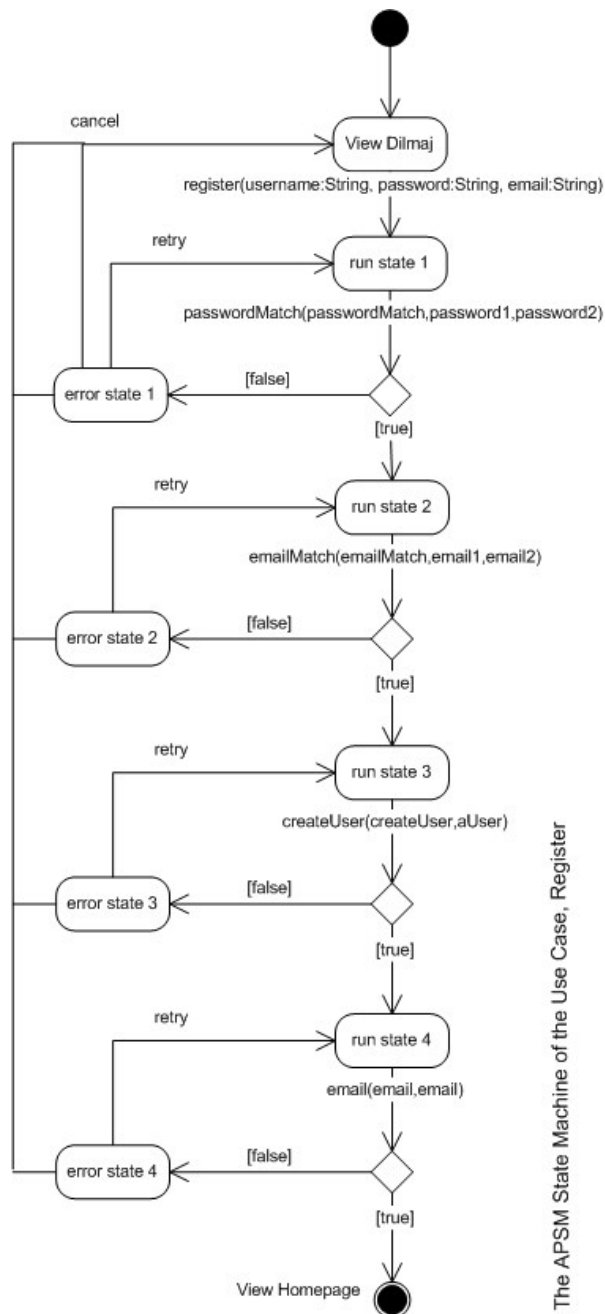
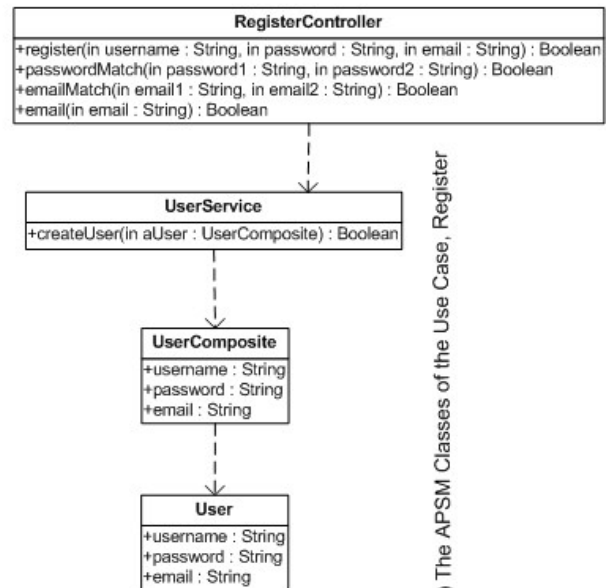


Figure 111 The Use Case, *Register*

Figure 112 shows the *apsm* of the use case, *Register*.



(a) The APSM State Machine of the Use Case, Register



(b) The APSM Classes of the Use Case, Register

Figure 112 The *apsm* of the Use Case, *Register*

B.15 Change Sort Criteria

This use case shown in Figure 112 changes a table sorting criteria. The new sort criterion is stored in a session variable, *Sort Criteria*. Figure 113

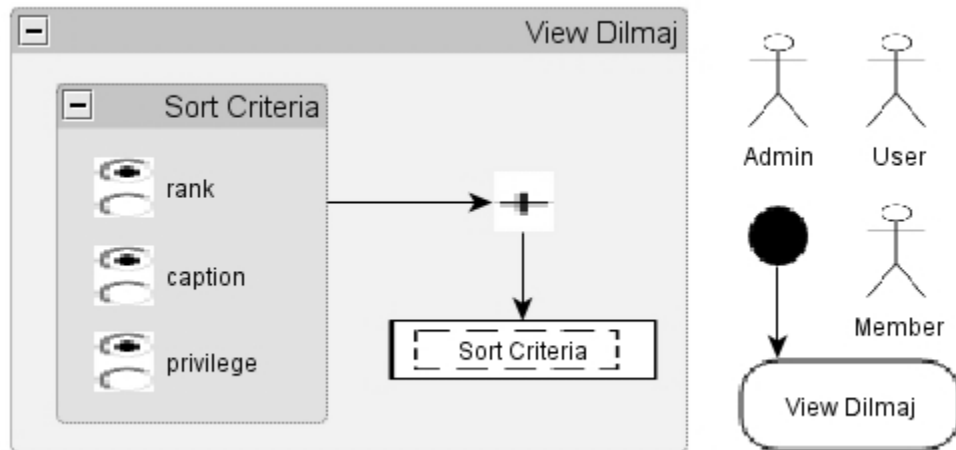


Figure 113 The Use Case, *Change Sort Criteria*

Figure 114 shows the *apsm* of the use case, *Change Sort Criteria*.

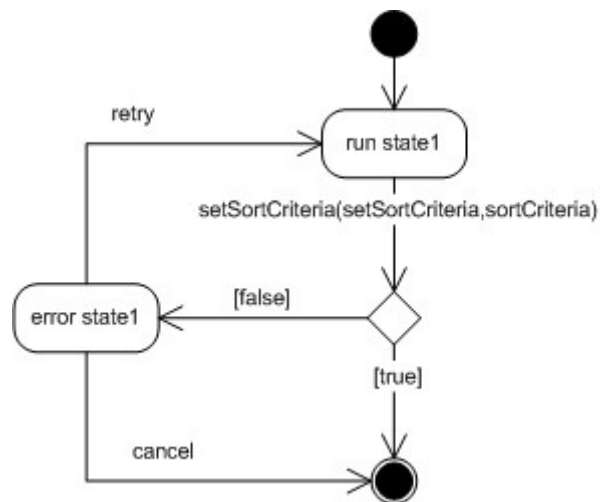


Figure 114 The *apsm* of the Use Case, *Change Sort Criteria*

B.16 View Use Case

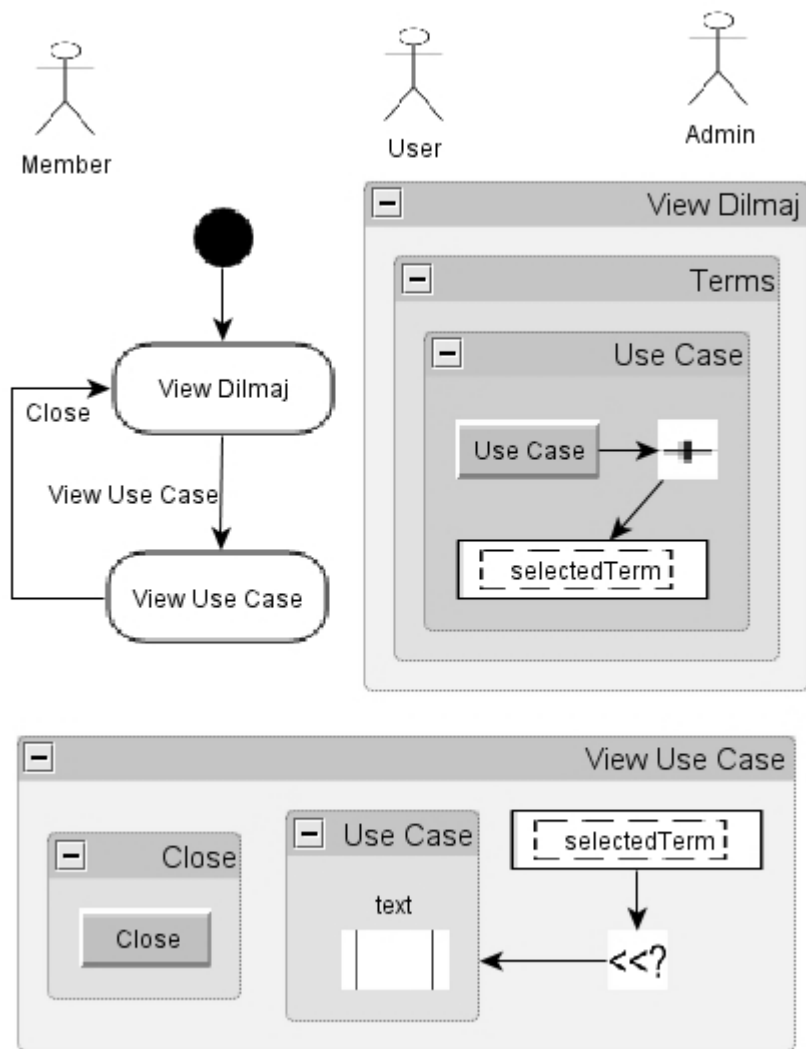
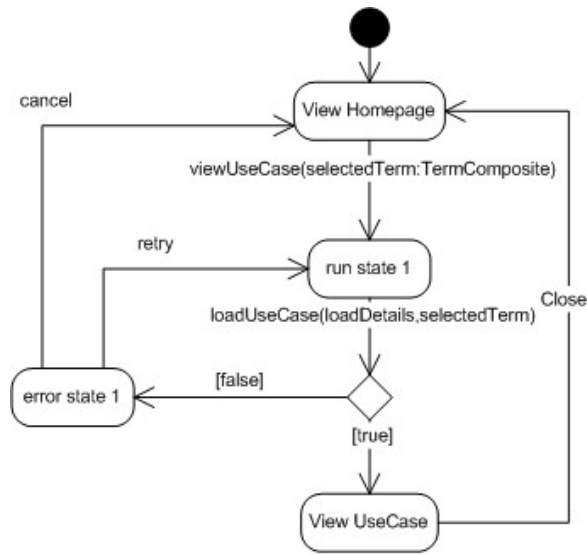
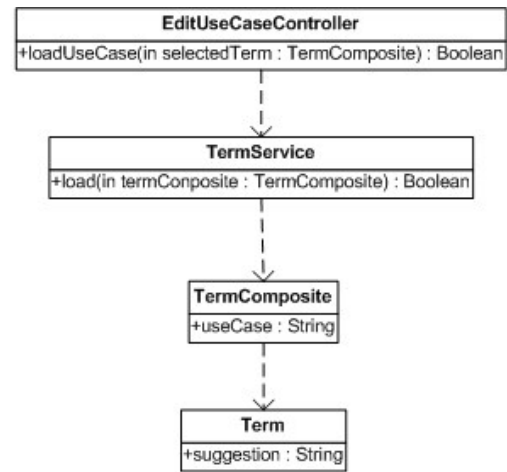


Figure 115 The Use Case, *View Use Case*

This use case shows the use case of every term.



(a) The APSM State Machine of the Use Case, View Use Case



(b) The APSM Classes of the Use Case, View Use Case

Figure 116 The *apsm* of the Use Case, *View Use Case*

B.17 Refine Area

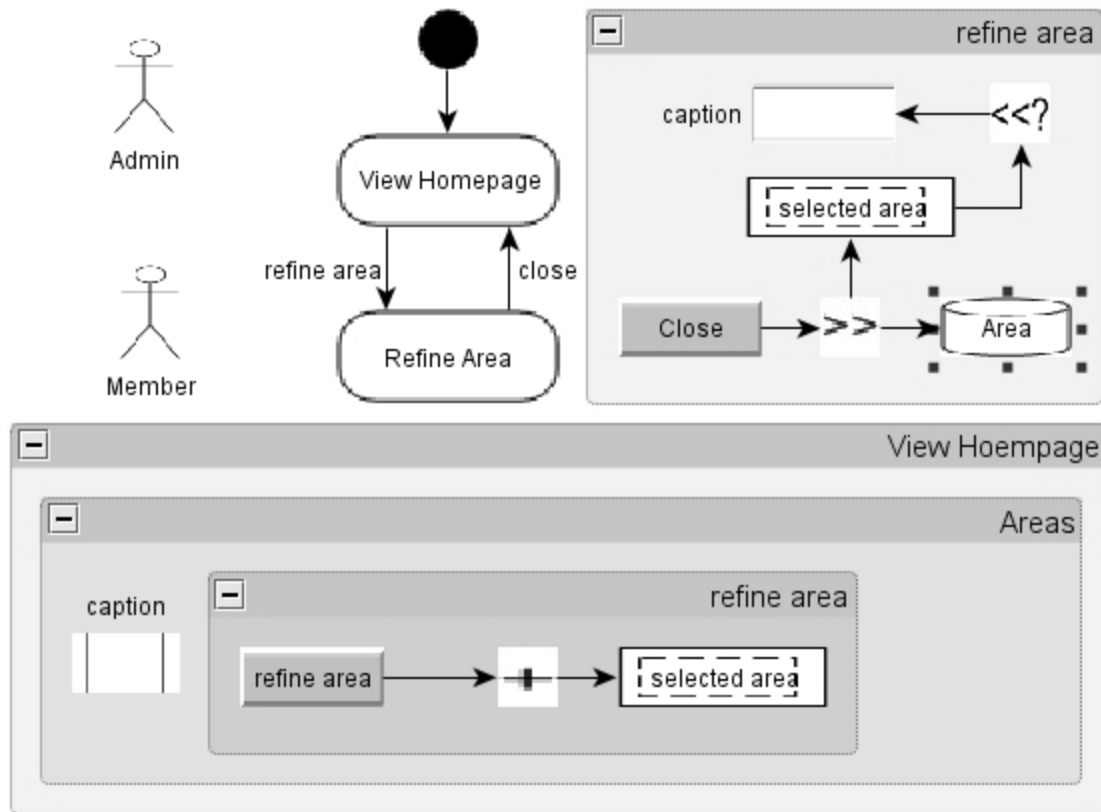


Figure 117 The Use Case, *Refine Area*

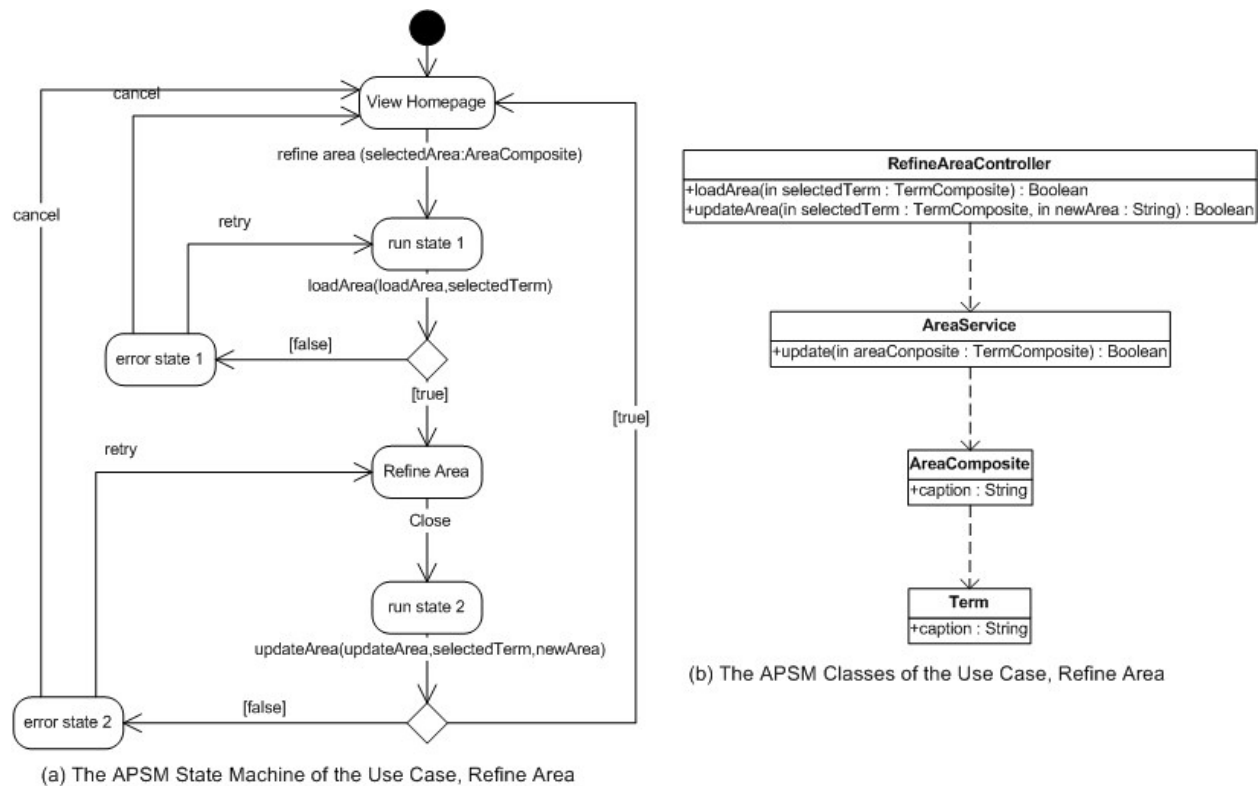


Figure 118 The *apsm* of the Use Case, *Refine Area*

B.18 Suggest a Term

Figure 119 shows a use case that adds a new term to the database. The UI prototype of this use case includes three different operations for adding a term to database, variable and both tables. As Figure 120 shows, the *apsm* version of the use case *Suggest a Term* includes sequencing calls to three different controller operations of the controller, *SuggestTermController*.

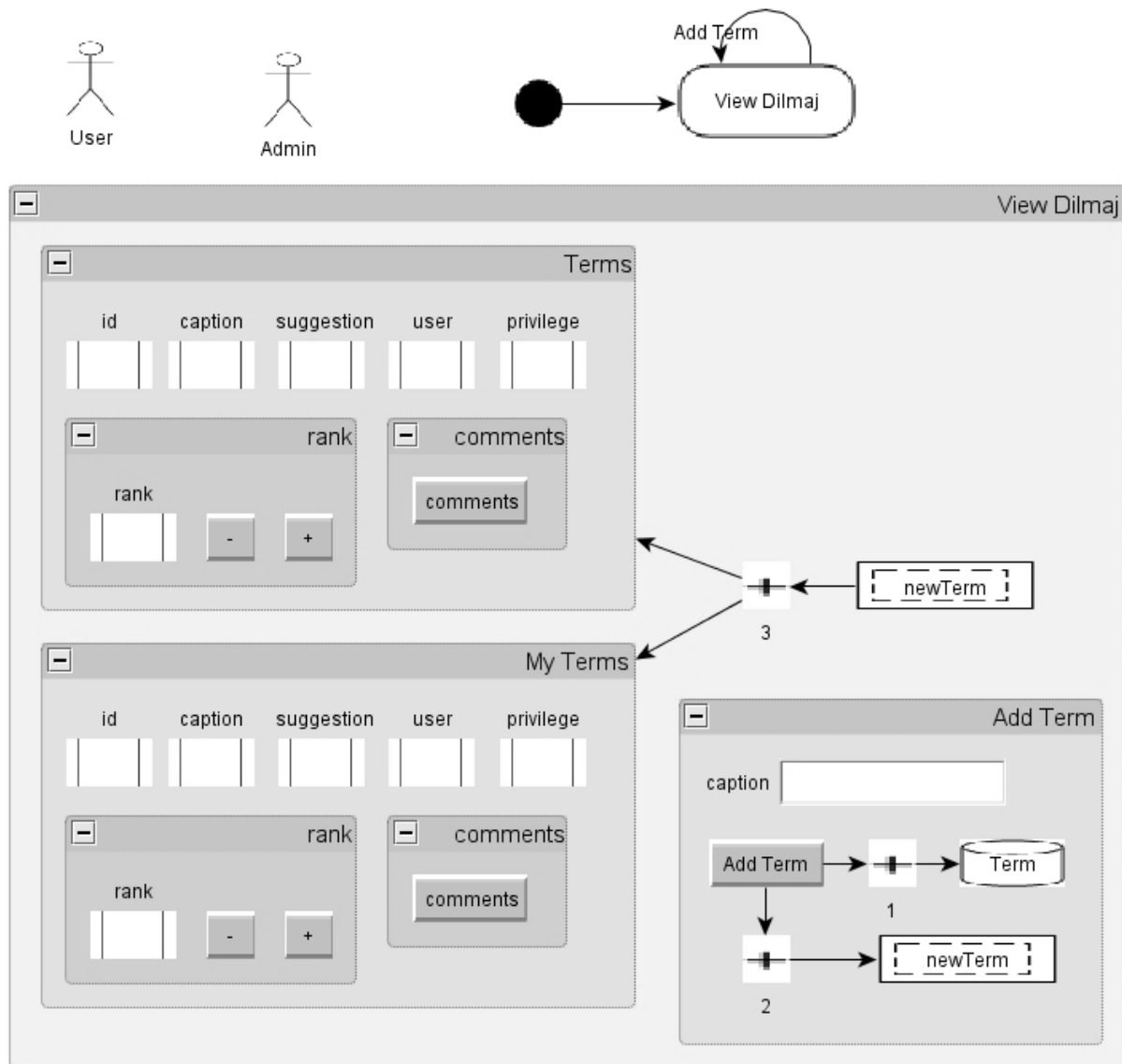


Figure 119 The Use Case, *Suggest a Term*

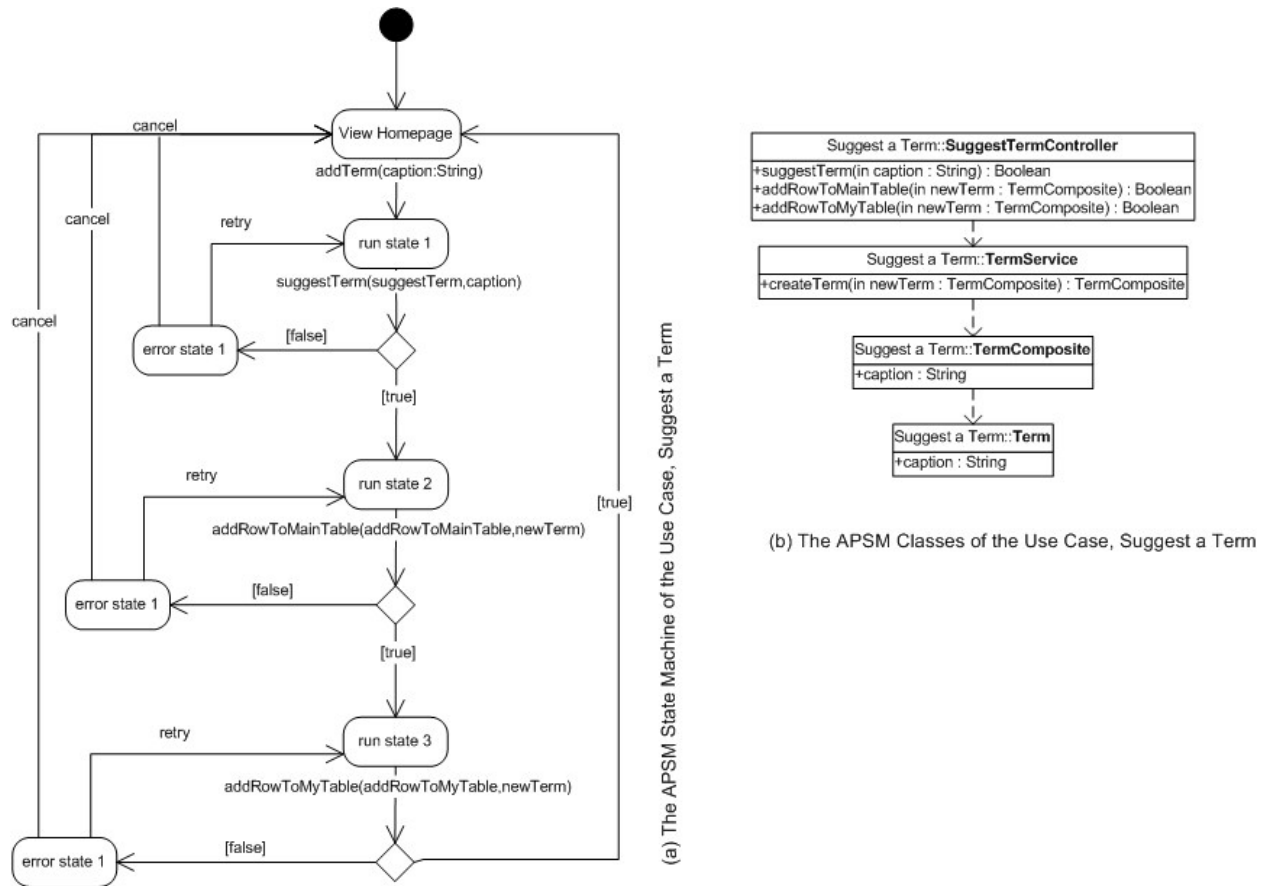


Figure 120 The *apsm* of the Use Case, *Suggest a Term*

B.19 View Online Users

Figure 121 shows the use case, *View Online Users*. This use case refers to the variable, *Online Users* in order to retrieve the list of online users and update the corresponding table. This variable is updated every time a member logs in/out. Figure 122 shows the *apsm* of the use case, *View Online Users*. One can observe that the variable *onlineUsers* is passed through transitions.

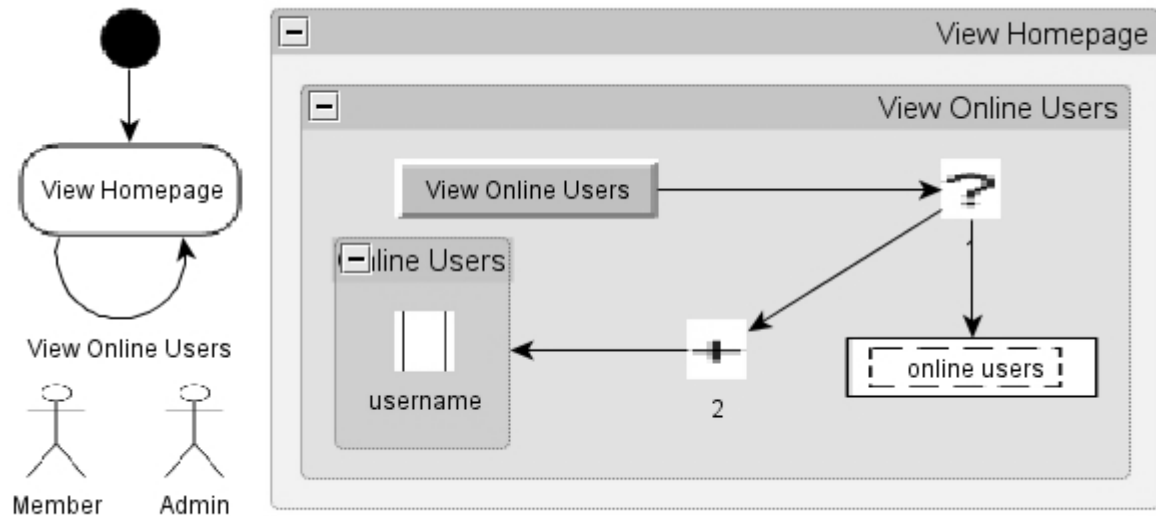


Figure 121 The Use Case, *View Online Users*

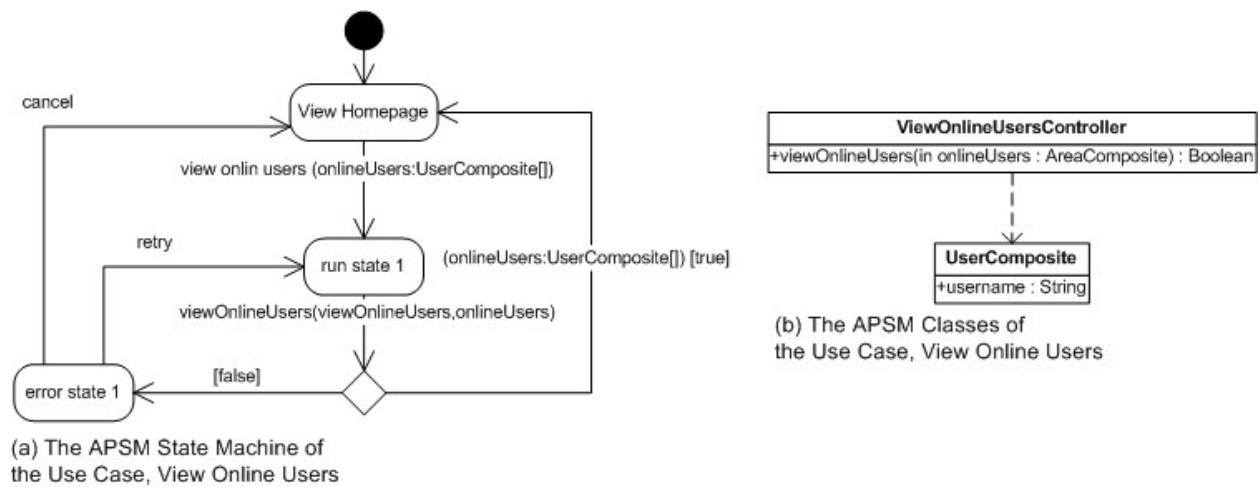


Figure 122 The *apsm* of the Use Case, *View Online Users*

B.20 Chat

Users may chat with other online users through a chat section as shown in Figure 123.

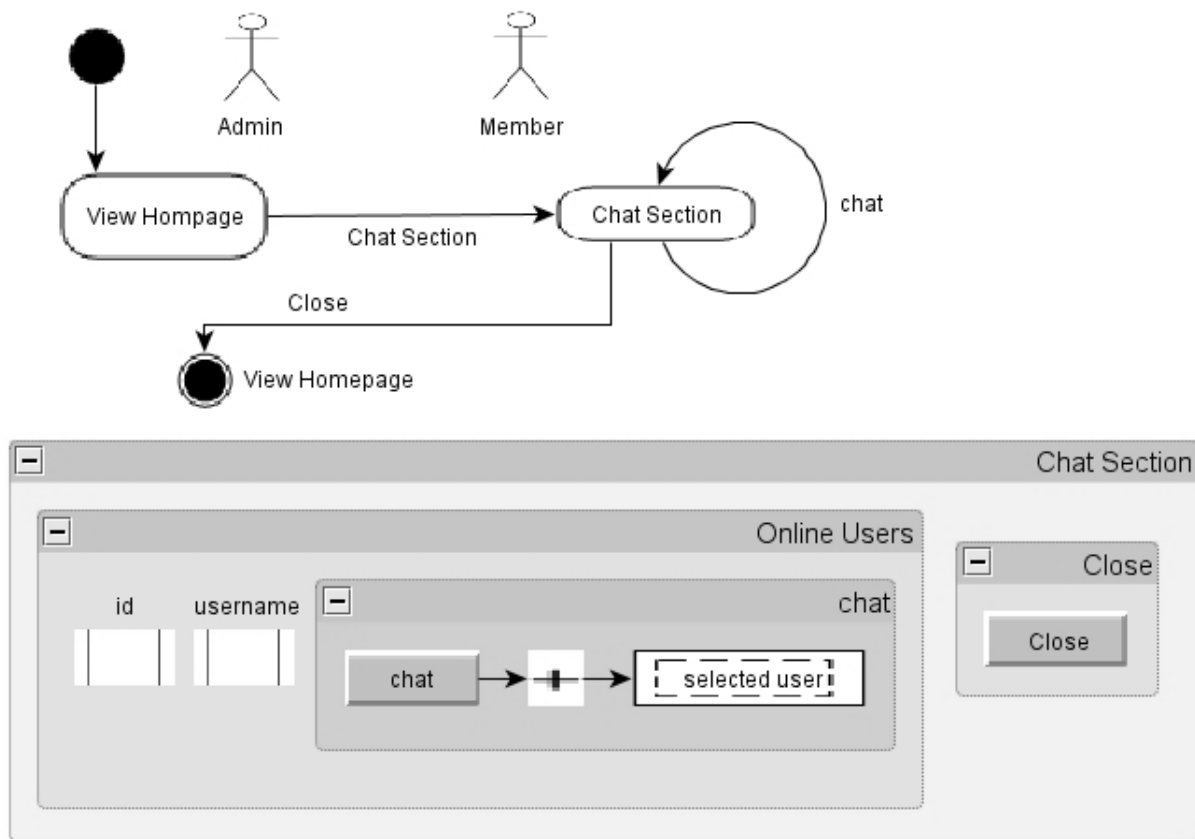


Figure 123 The Use Case, *Chat*

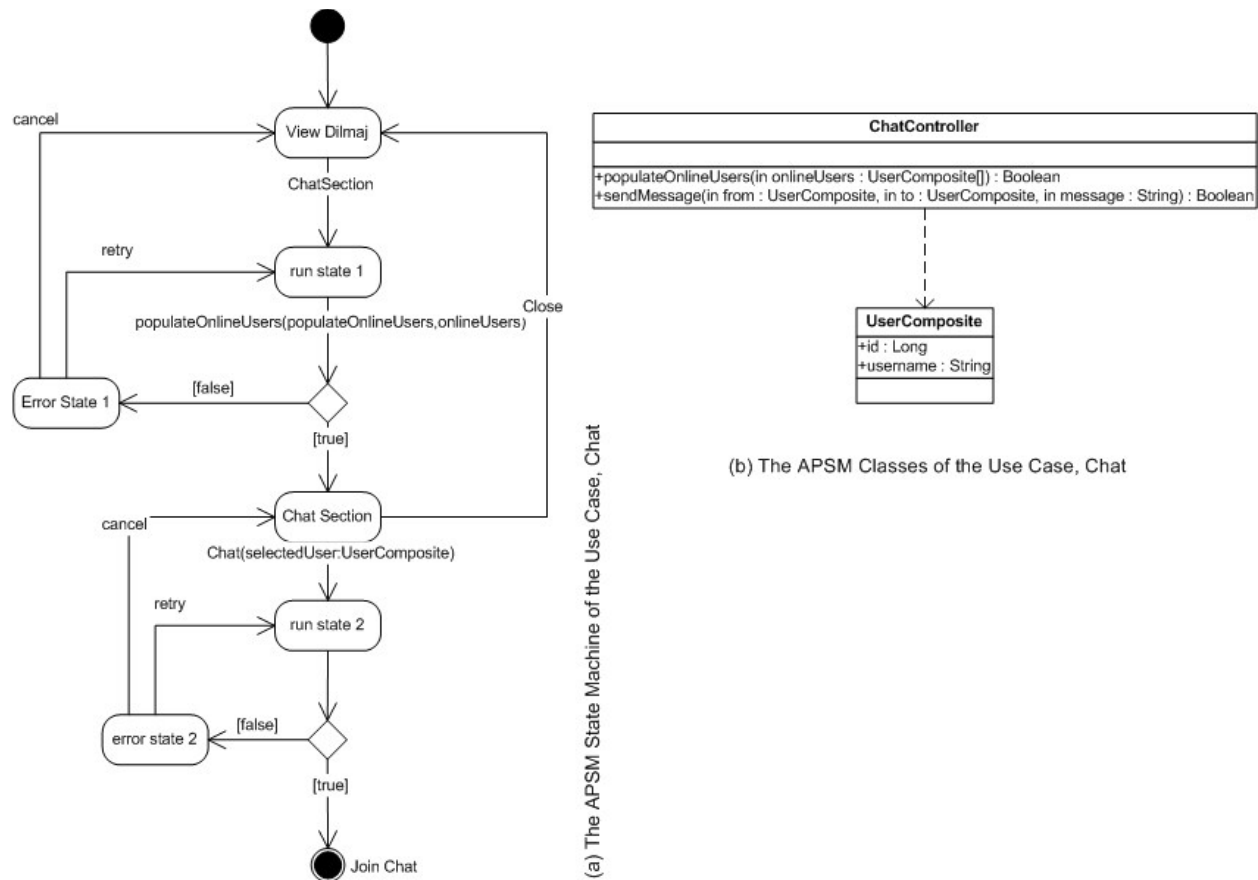
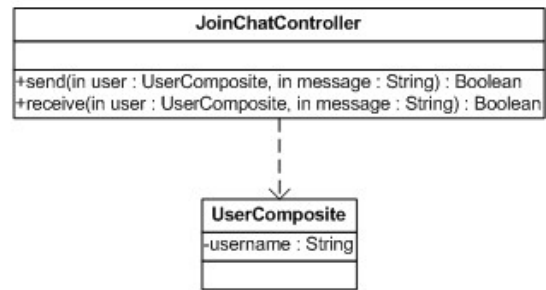
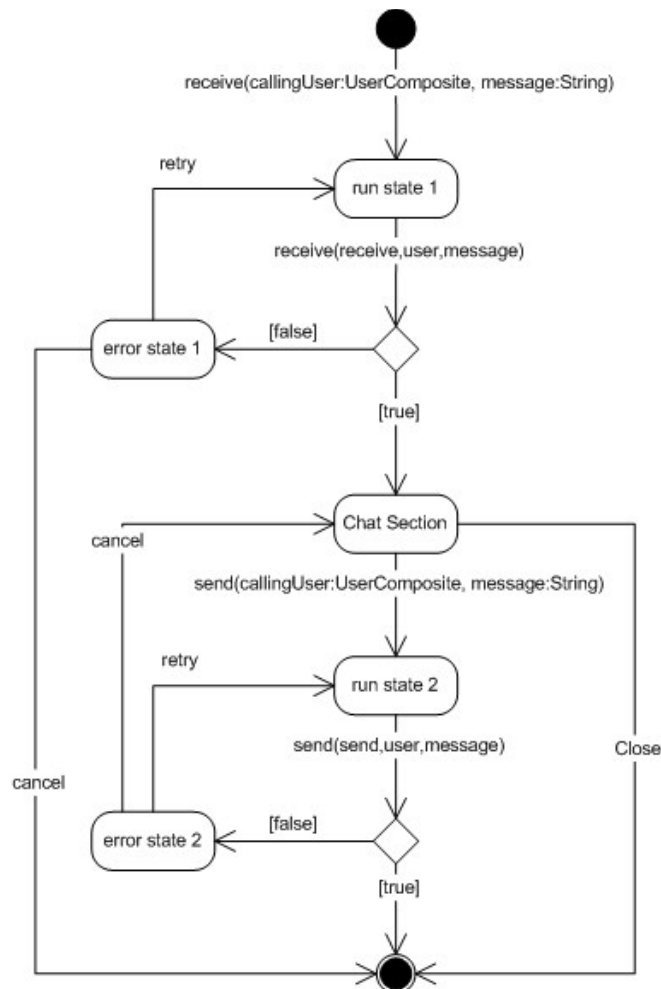


Figure 124 The *apsm* of the Use Case, *Chat*

B.21 Join Chat

Figure 125 shows the use case that handles the case when a user receives a message from another user.



(b) The APSM Classes of the Use Case, Join Chat

(a) The APSM State Machine of the Use Case, Join Chat

Figure 125 The *apsm* of the *Join Chat*

Figure 126 shows the final class model resulted from integrating all the above *apsm* classes. Note that the integration is automatically performed by the automated transformations and different strategies used for guaranteeing the uniqueness of elements.

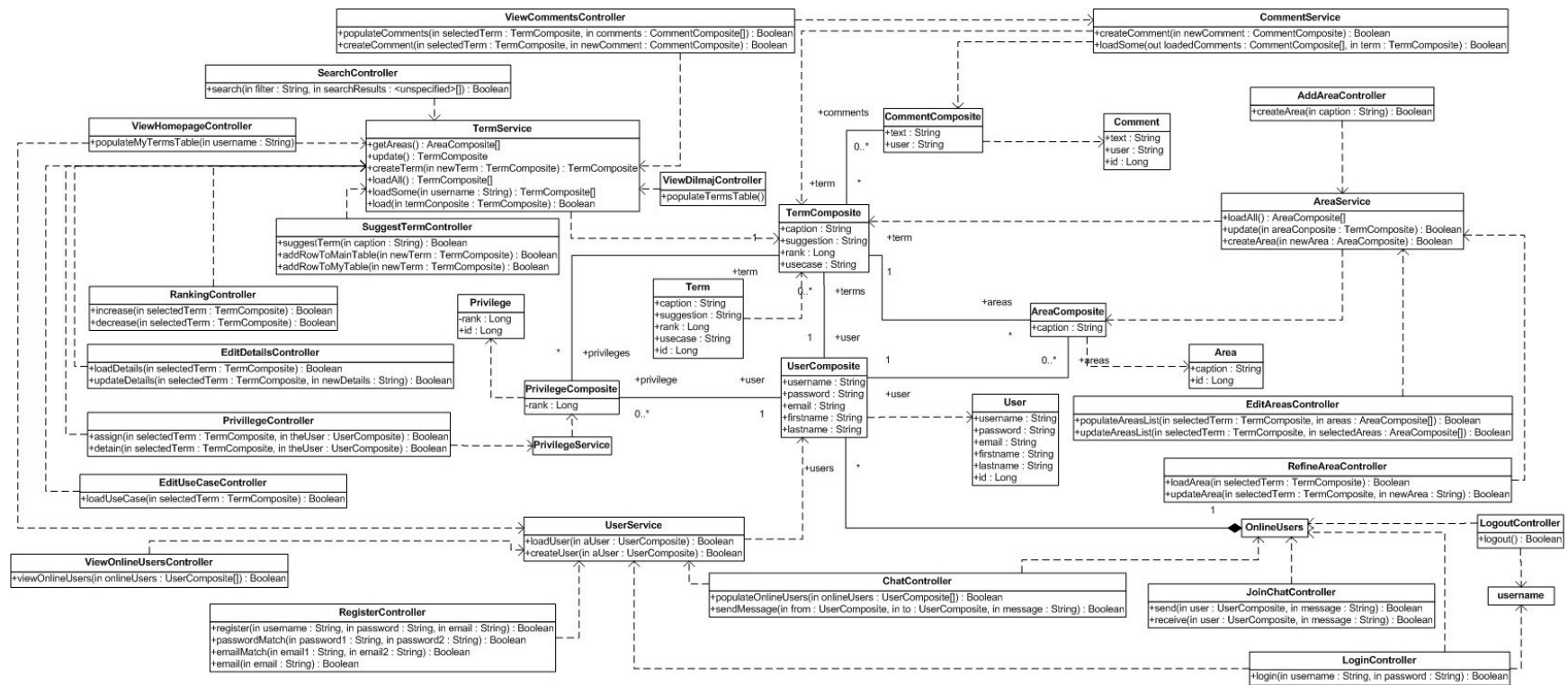


Figure 126 The *apsm* of the application *Dilmaj*

B.22 Summary

In this appendix, we presented the *pim* and *apsm* of the case study Dilmaj. Dilmaj is developed using Google Web Toolkit, Java Data Objects, Java Mail API and Google Application Engine. A stable working application can be found in [Dilmaj2010] and the under-development application can be found in [Dilmaj2011]. Dilmaj is open-source, the code of the stable version – that is the one created using the models in this appendix - is found in [Sokhan2010] and the under-development code – the manually customized version - is found in [Sokhan2011].

Appendix C. AMS CASE STUDY

AMS is an Account Management System, developed to replace an existing system with a web-based version. In this appendix, the *pim* and the *apsm* of the AMS are shown. We start by reviewing the *pim*, then we present the *apsm*.

C.1 AMS *pim*

C.1.1 Add Admin Use Case

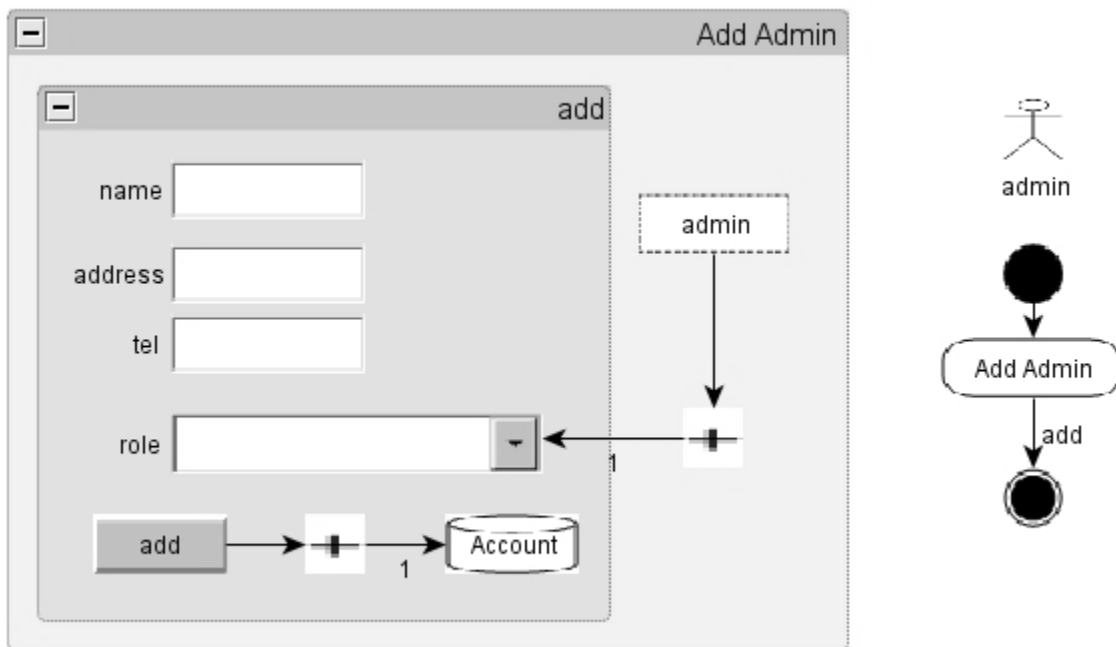


Figure 127 The *pim* of the *Add Admin* use case

C.1.2 Add New Bank Use Case

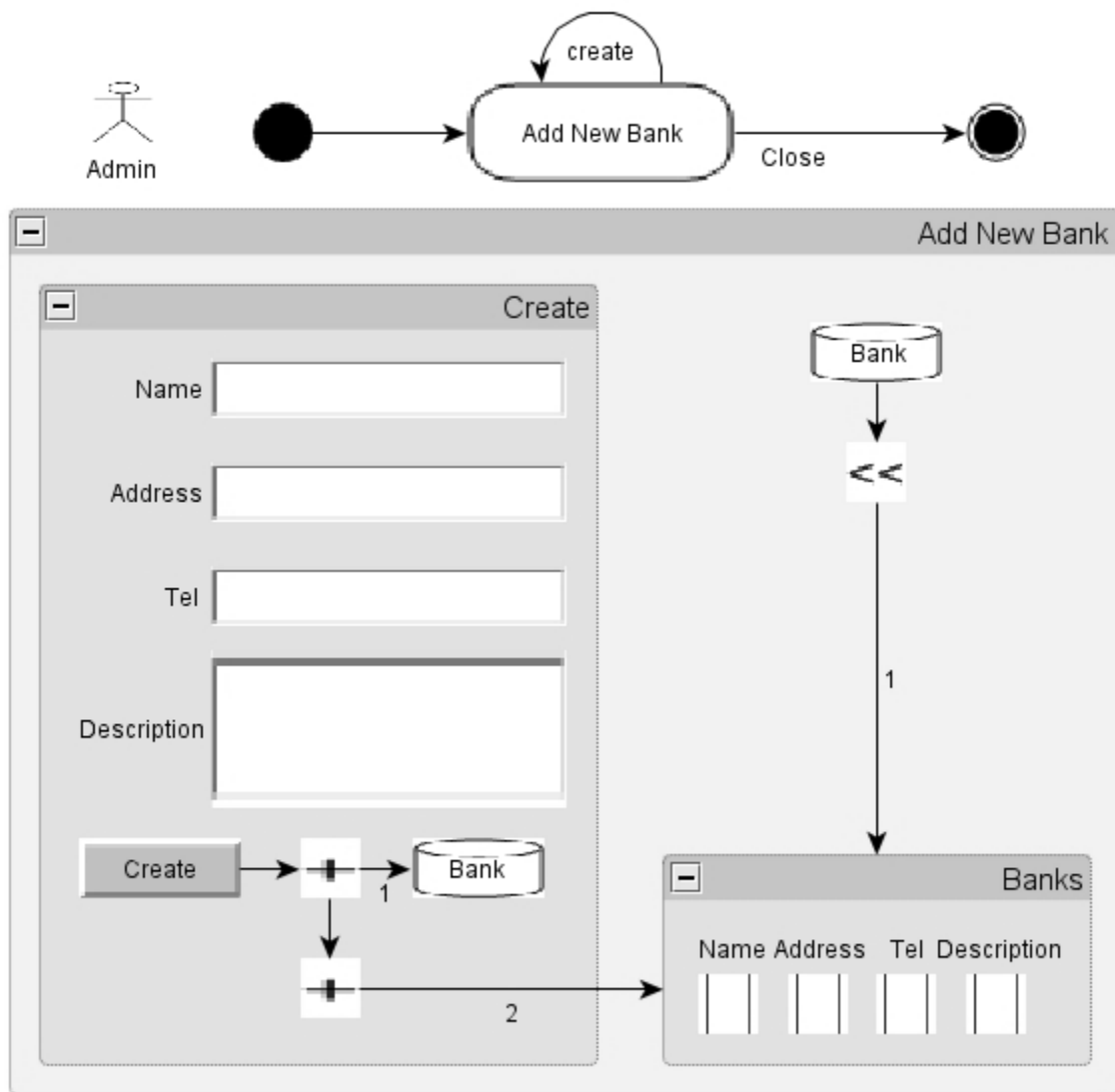


Figure 128 The *pim* of the *Add New Bank* use case

C.1.3 Add News

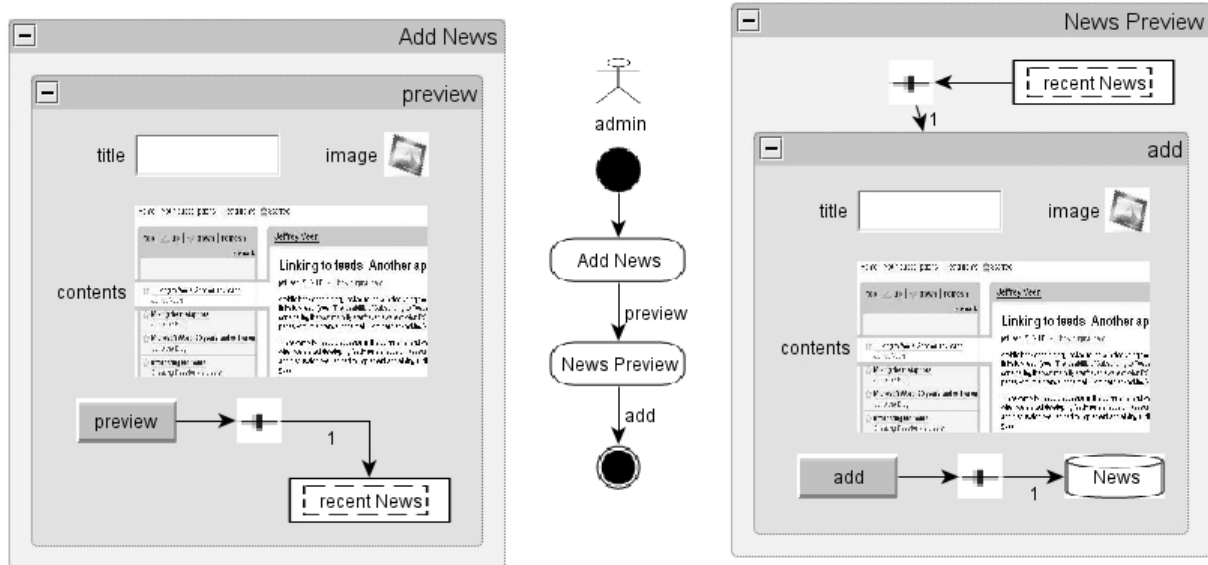


Figure 129 The *pim* of the *Add News* use case

C.1.4 Add Tutorial

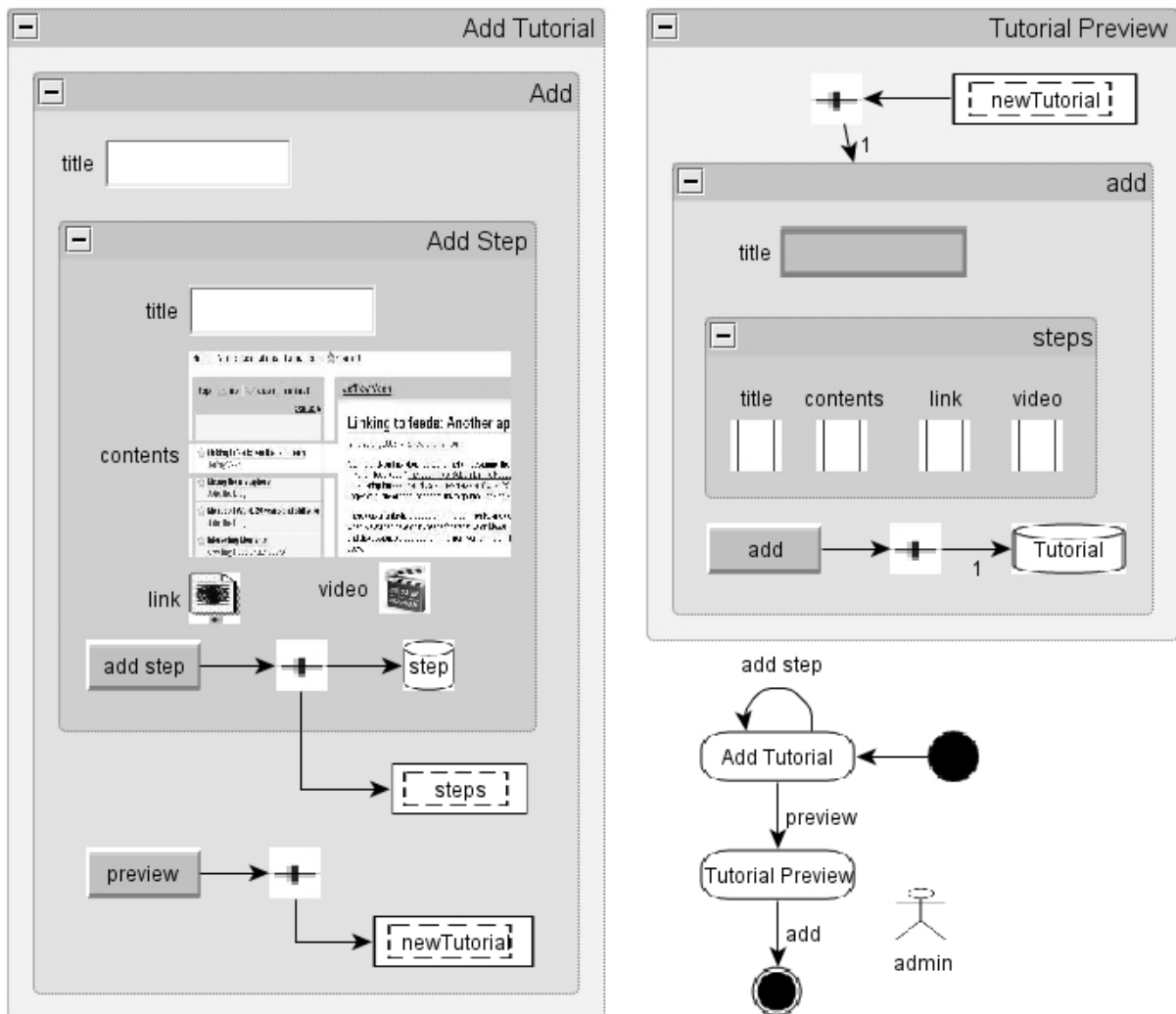


Figure 130 The *pim* of *Add Tutorial* use case

C.1.5 Add User

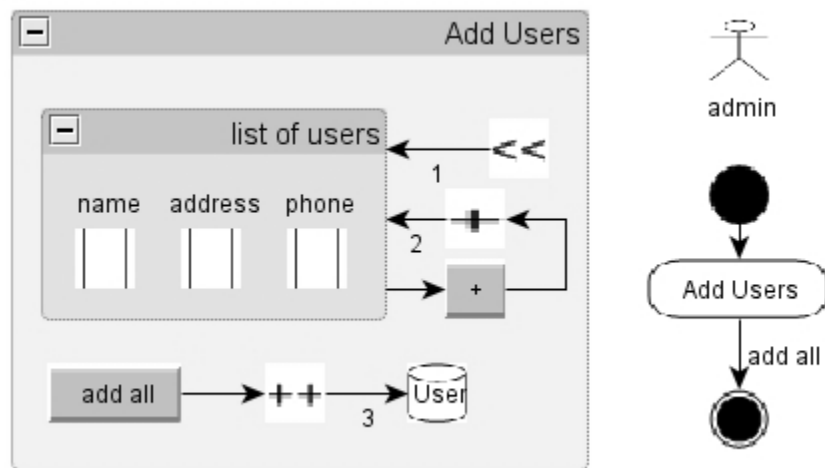


Figure 131 The *pim* of the *Add User* use case

C.1.6 Add Billing Rule

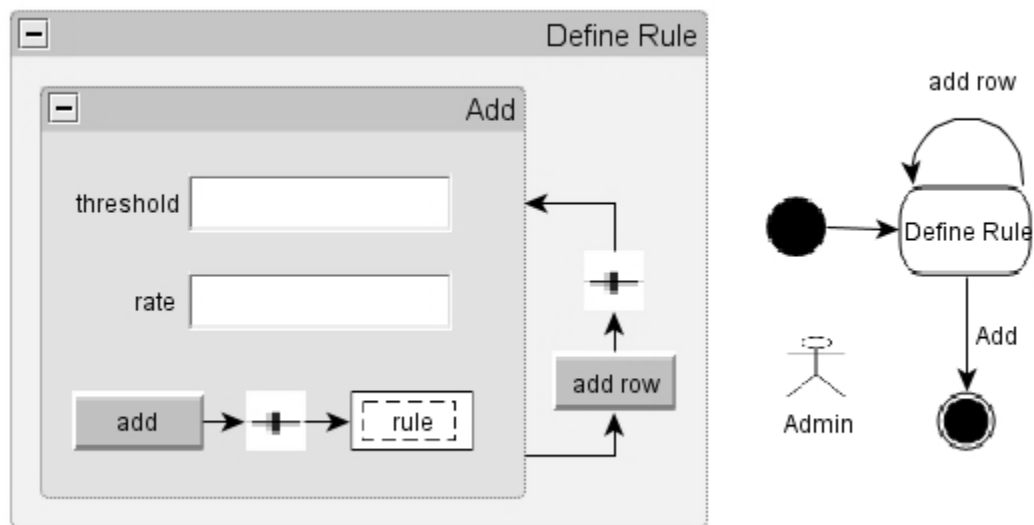


Figure 132 The *pim* of the *Add Billing Rule* use case

C.1.7 Address an Inquiry

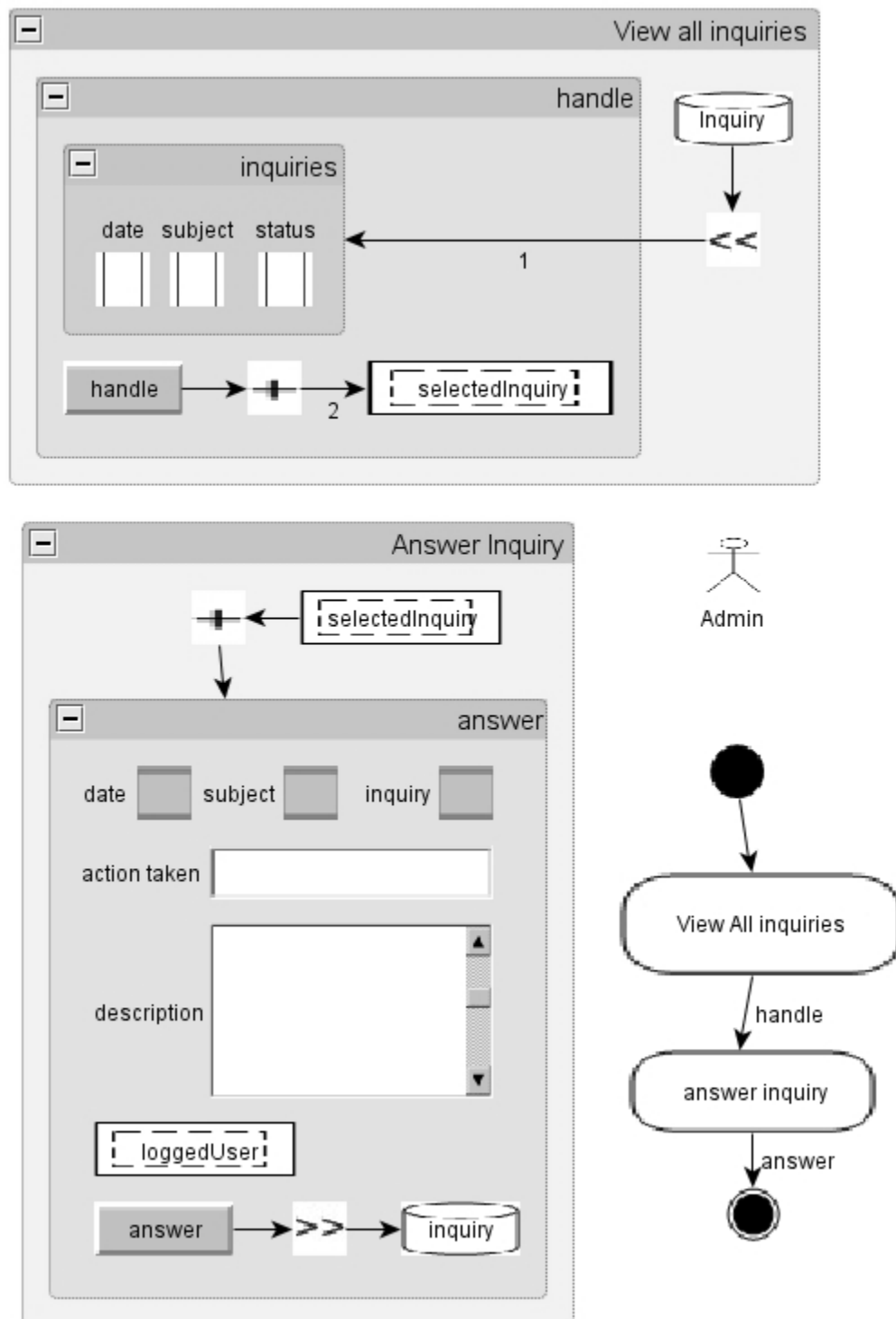


Figure 133 The *pim* of *Address an Inquiry* use case

C.1.8 Apply Penalty

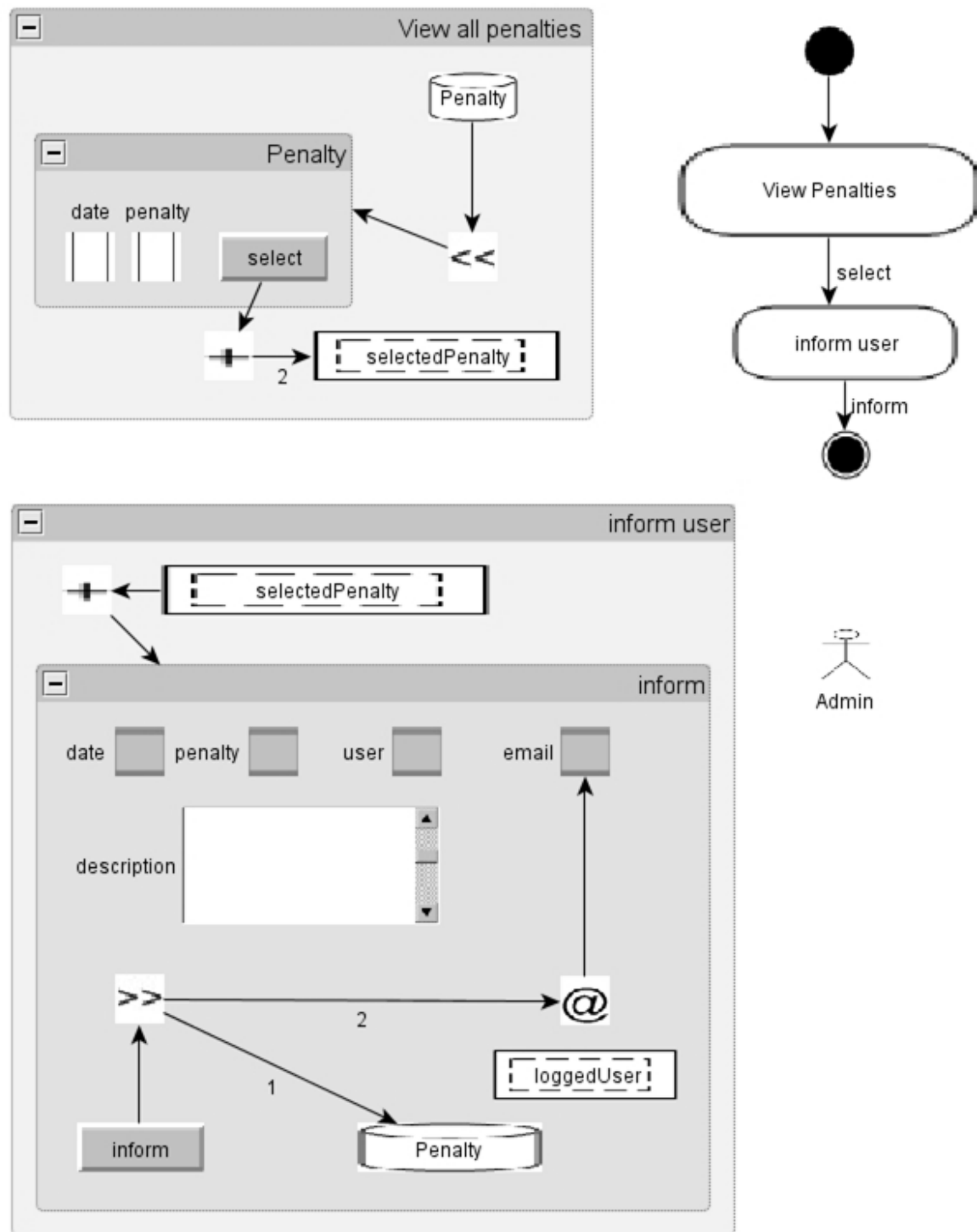


Figure 134 The *pim* of the *Apply Penalty* use case

C.1.9 Assign Account

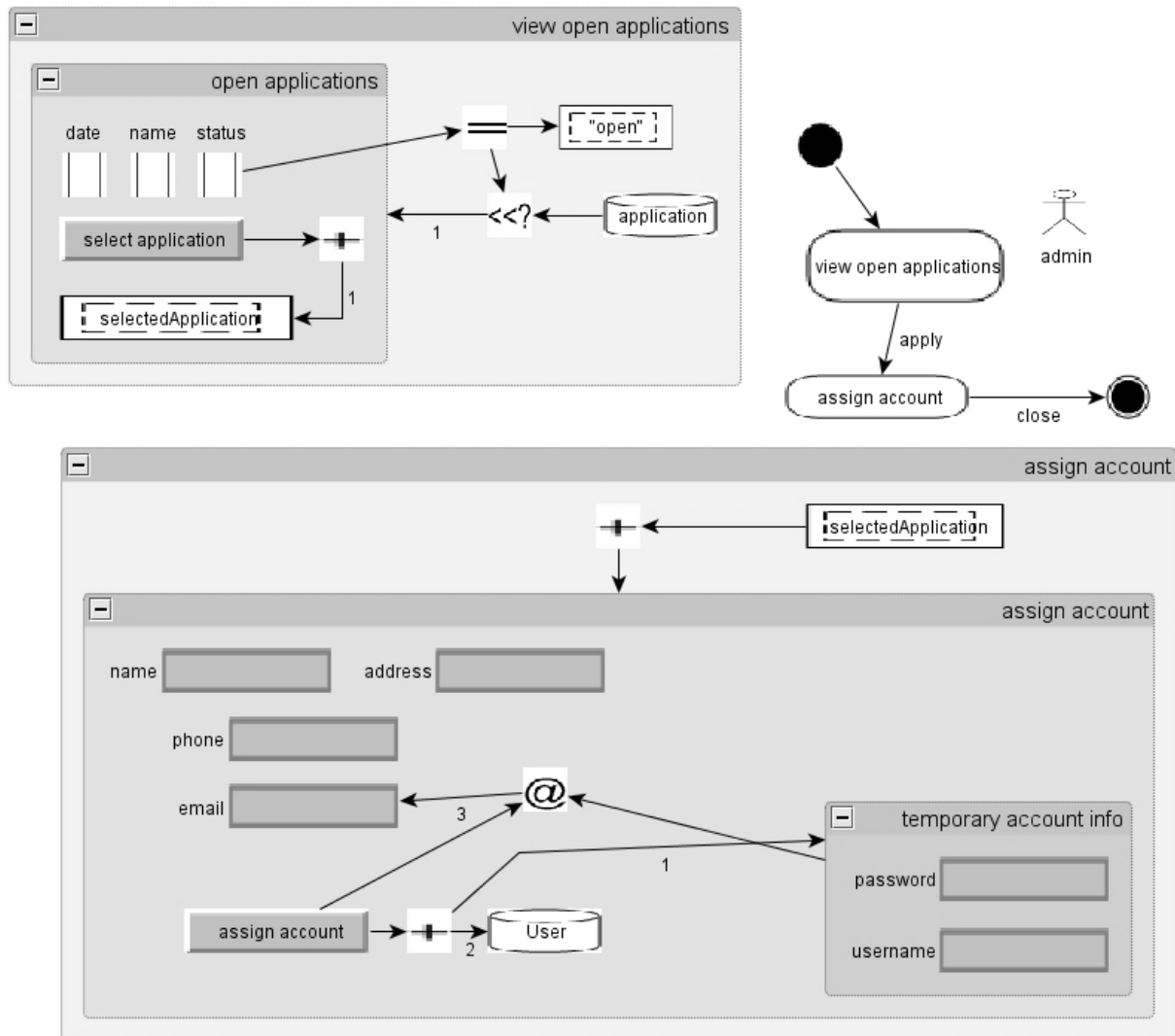


Figure 135 The *pim* of the *Assign Account* use case

C.1.10Cancel Instalment

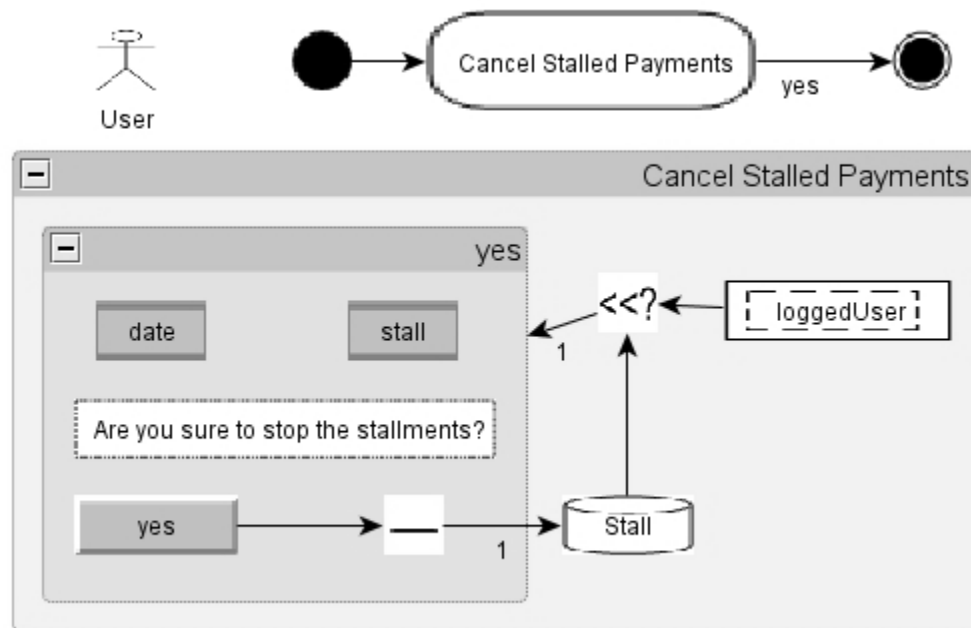


Figure 136 The *pim* of the *Cancel Instalment* use case

C.1.11 Confirm Account Request

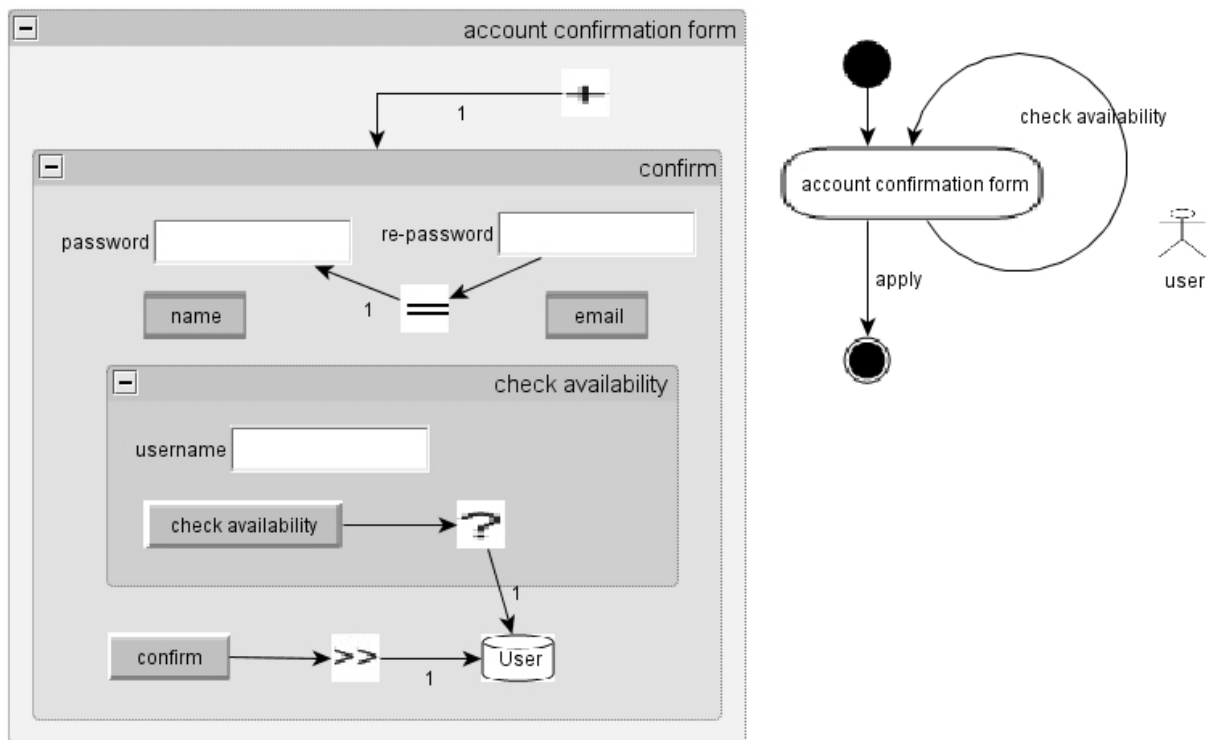


Figure 137 The *pim* of the *Confirm Account Request* use case

C.1.12 Confirm Instalment Offer

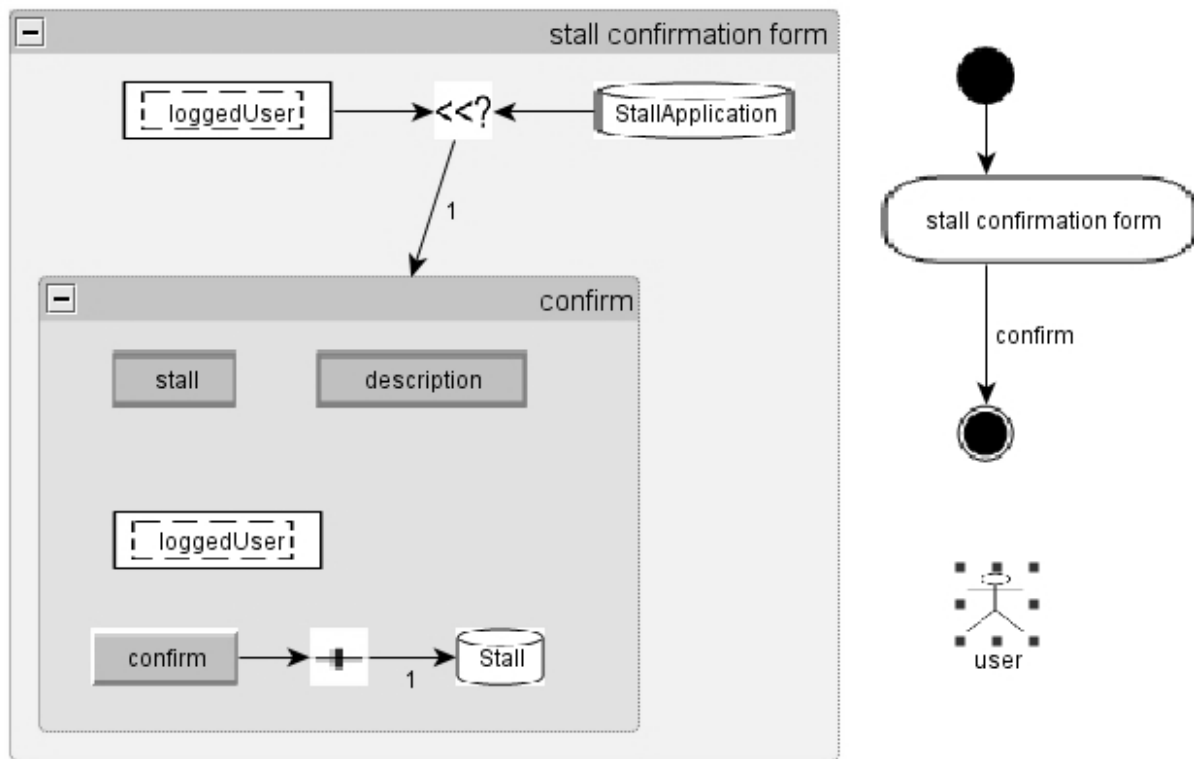


Figure 138 The *pim* of the *Confirm Instalment Offer* use case

C.1.13 Create Newsletter

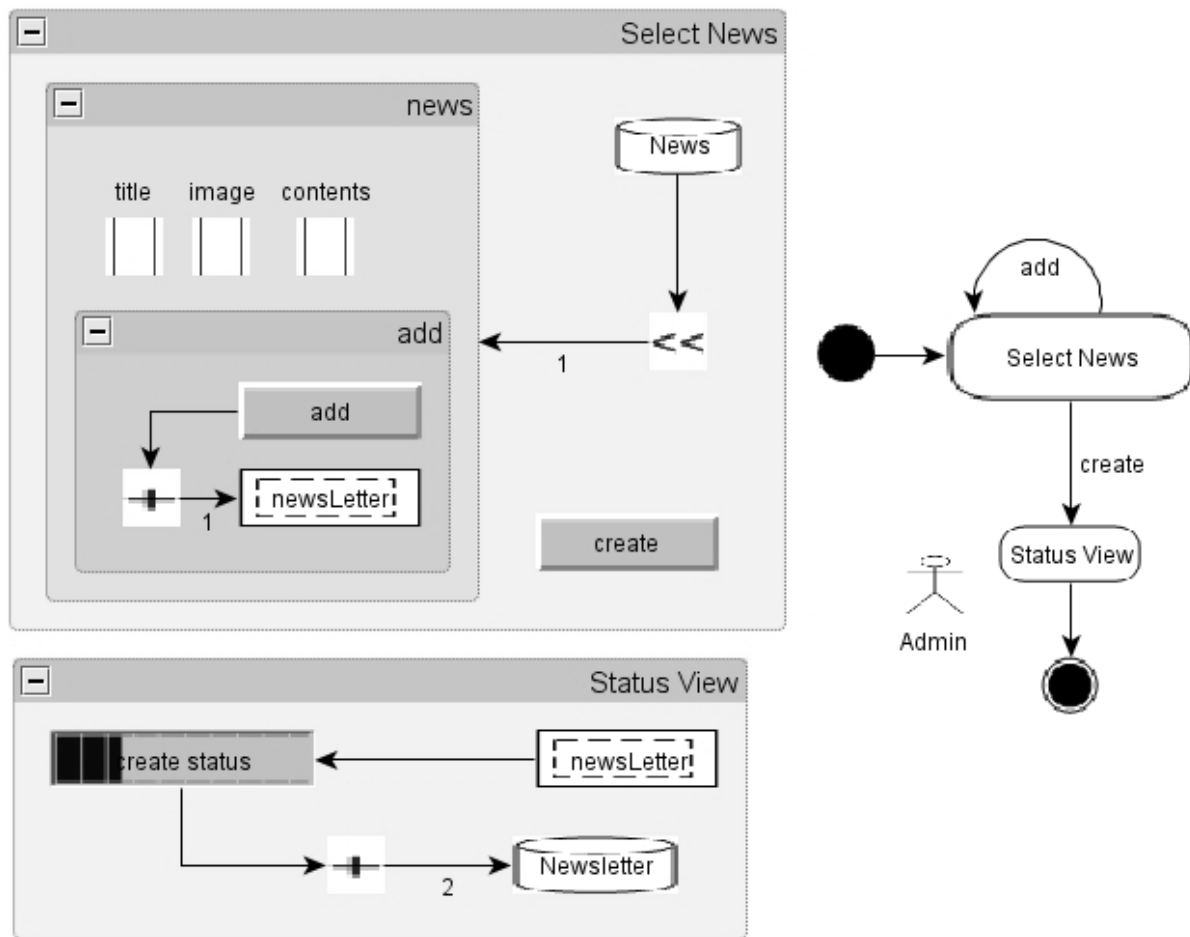


Figure 139 The *pim* of *Create Newsletter* use case

C.1.14 Edit Billing Rule

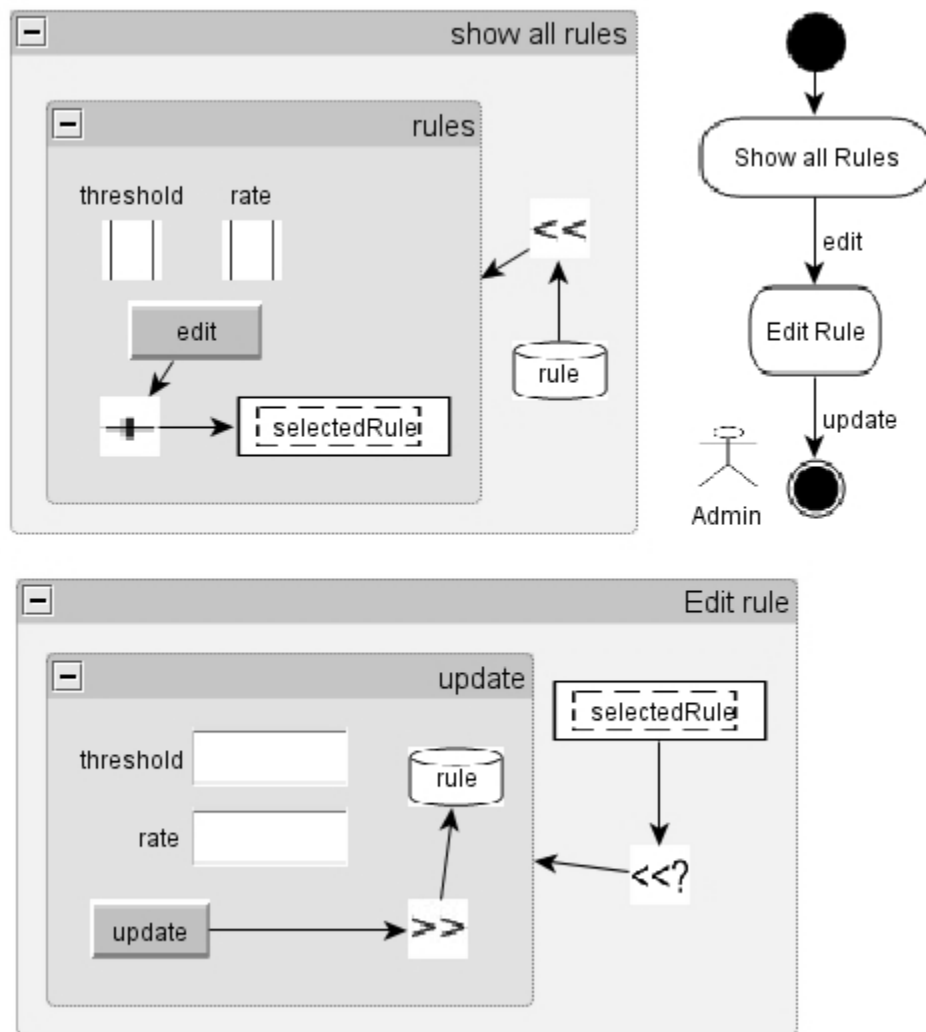


Figure 140 The *pim* of *Edit Rule* use case

C.1.15 Create Report

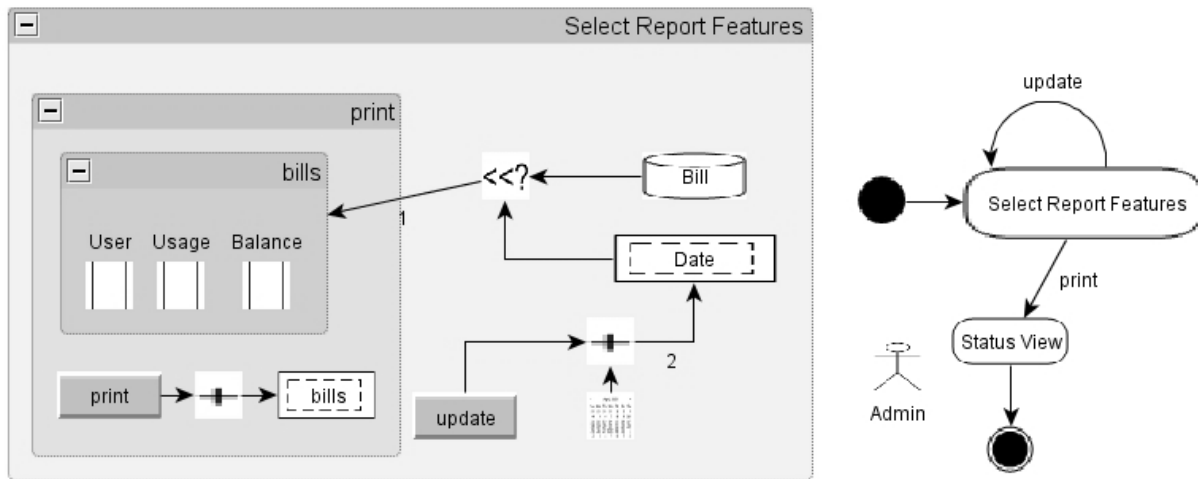


Figure 141 The *pim* of *Create Report* use case

C.1.16 Edit Personal Information

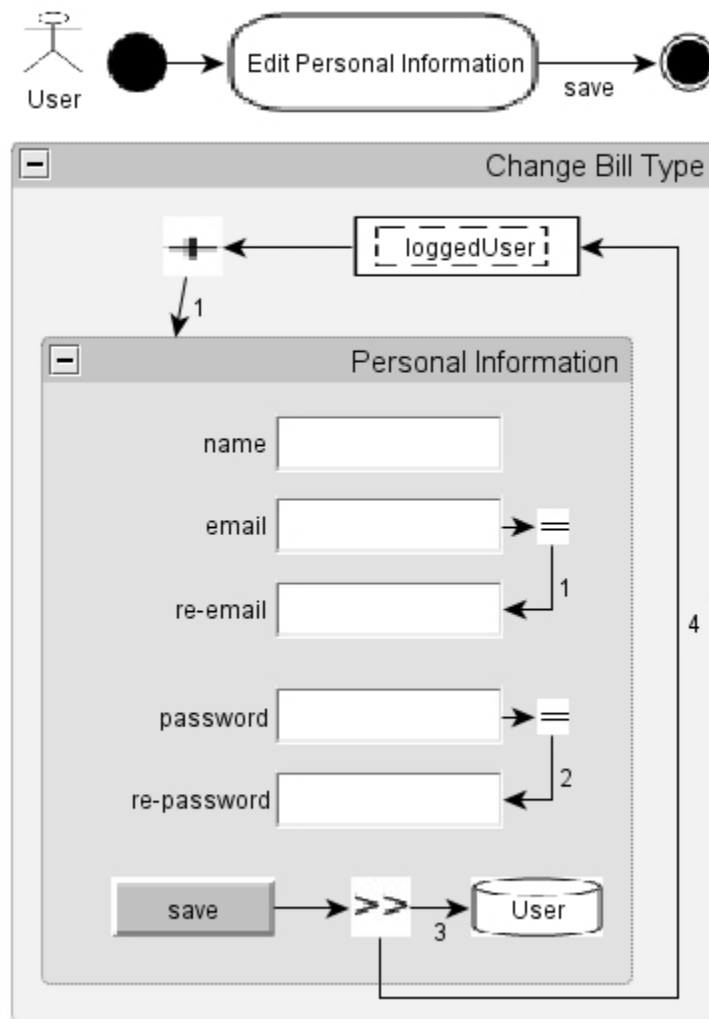


Figure 142 The *pim* of *Edit Personal Information* use case

C.1.17Handle Appointment

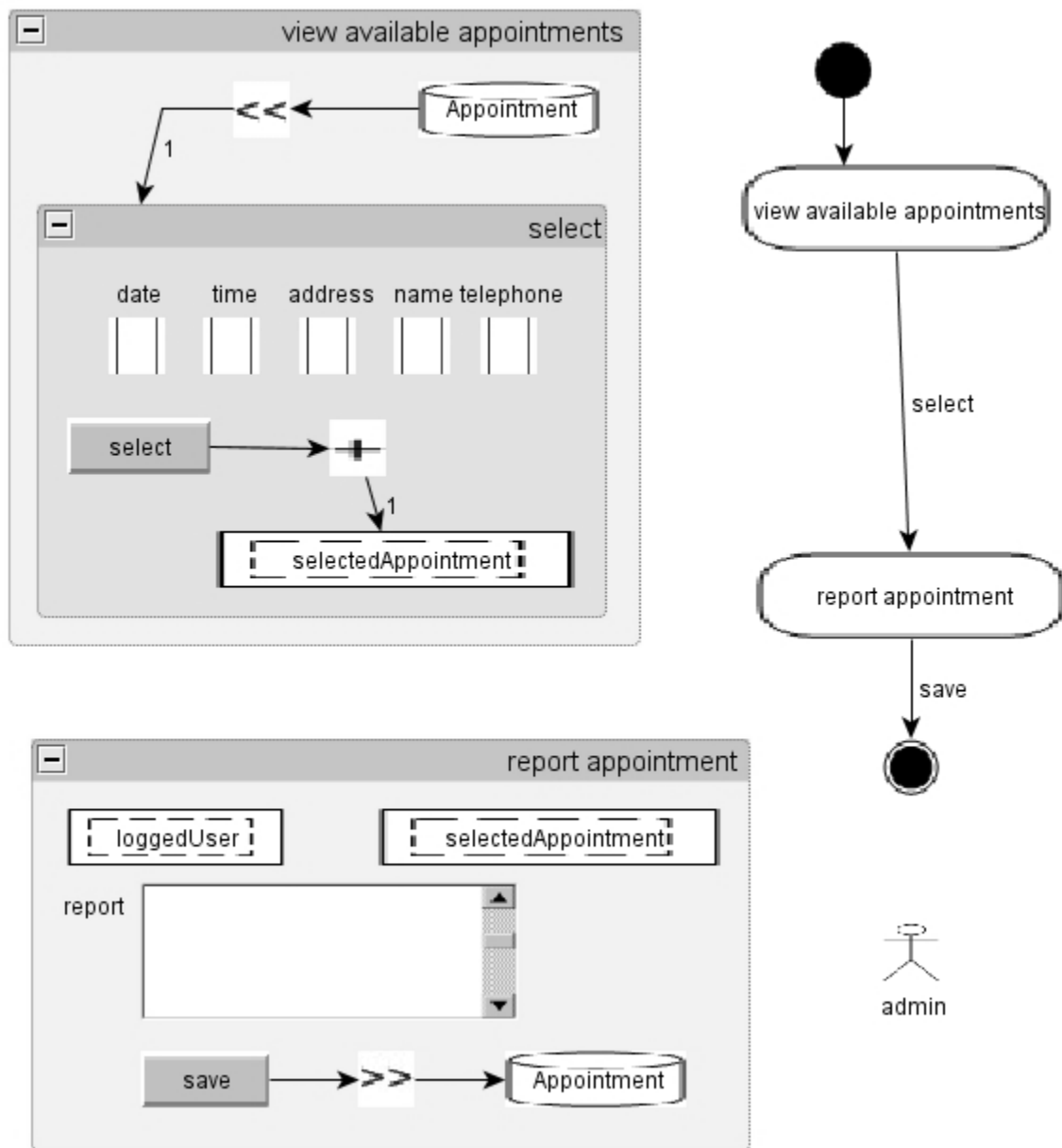


Figure 143 The *pim* of *Handle Appointment* use case

C.1.18 Inquire a Bill

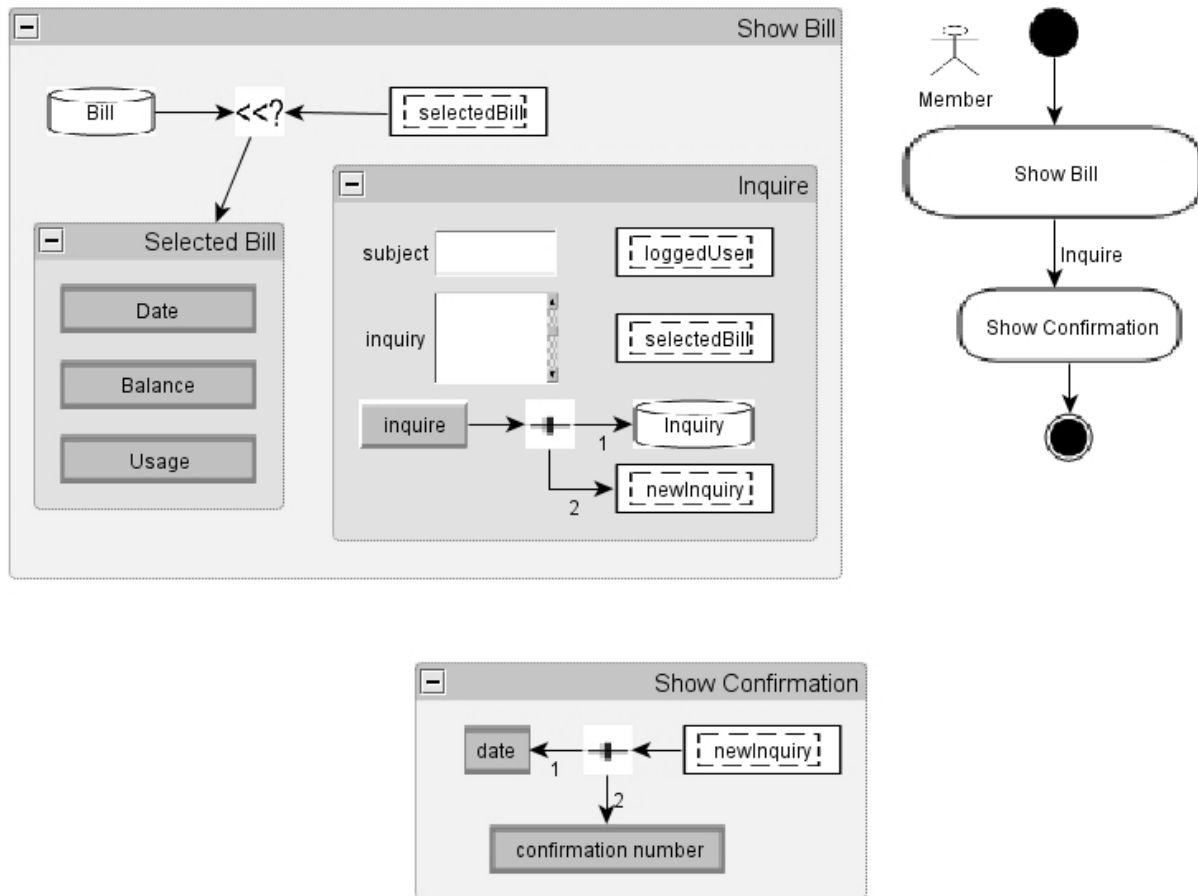


Figure 144 The *pim* of *Inquire a Bill* use case

C.1.19 Login

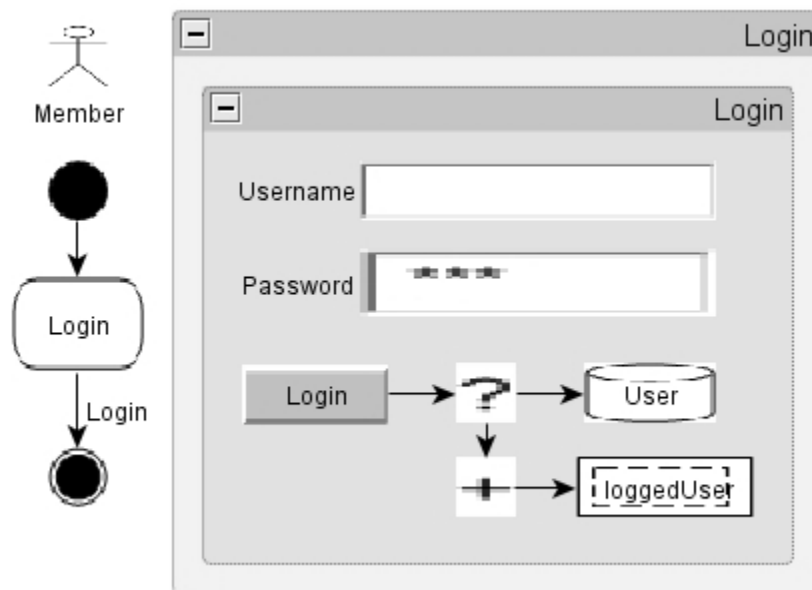


Figure 145 The *pim* of *Login* use case

C.1.20 Subscribe to Newsletter

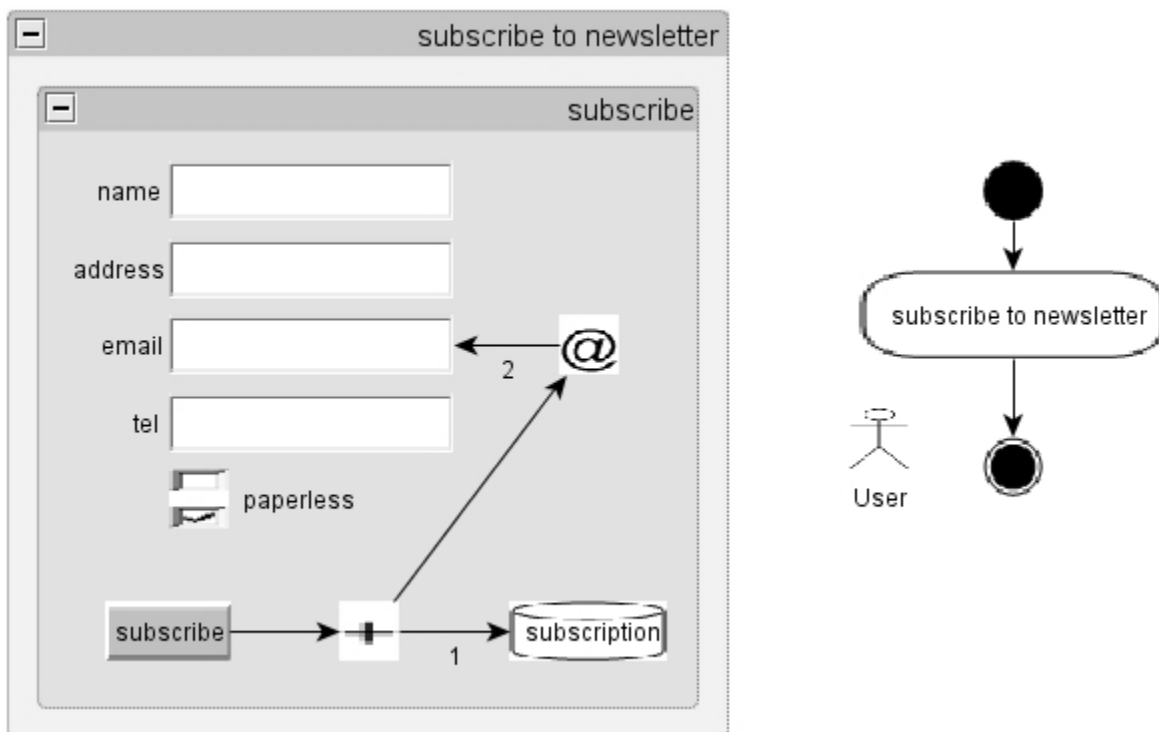


Figure 146 The *pim* of *Subscribe to Newsletter* use case

C.1.21 Unsubscribe from Newsletter

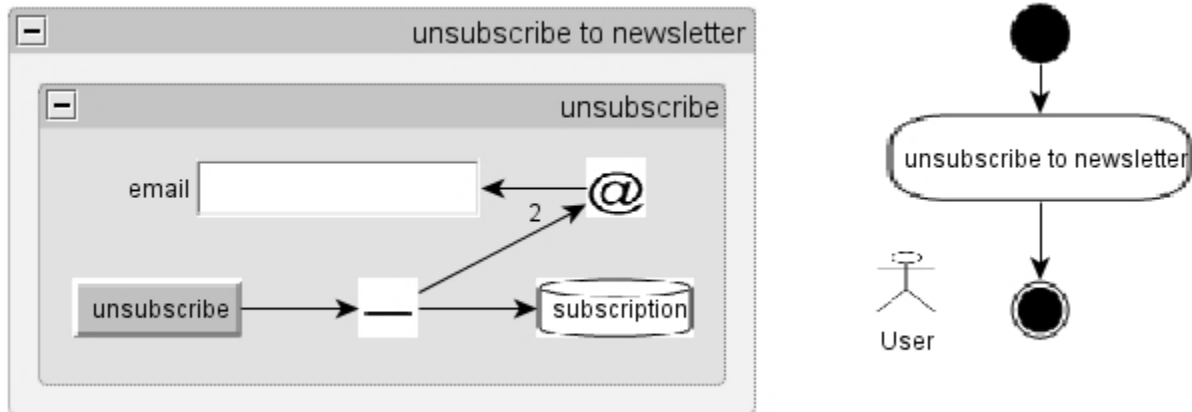


Figure 147 The *pim* of *Unsubscribe from Newsletter* use case

C.1.22 Open Newsletter

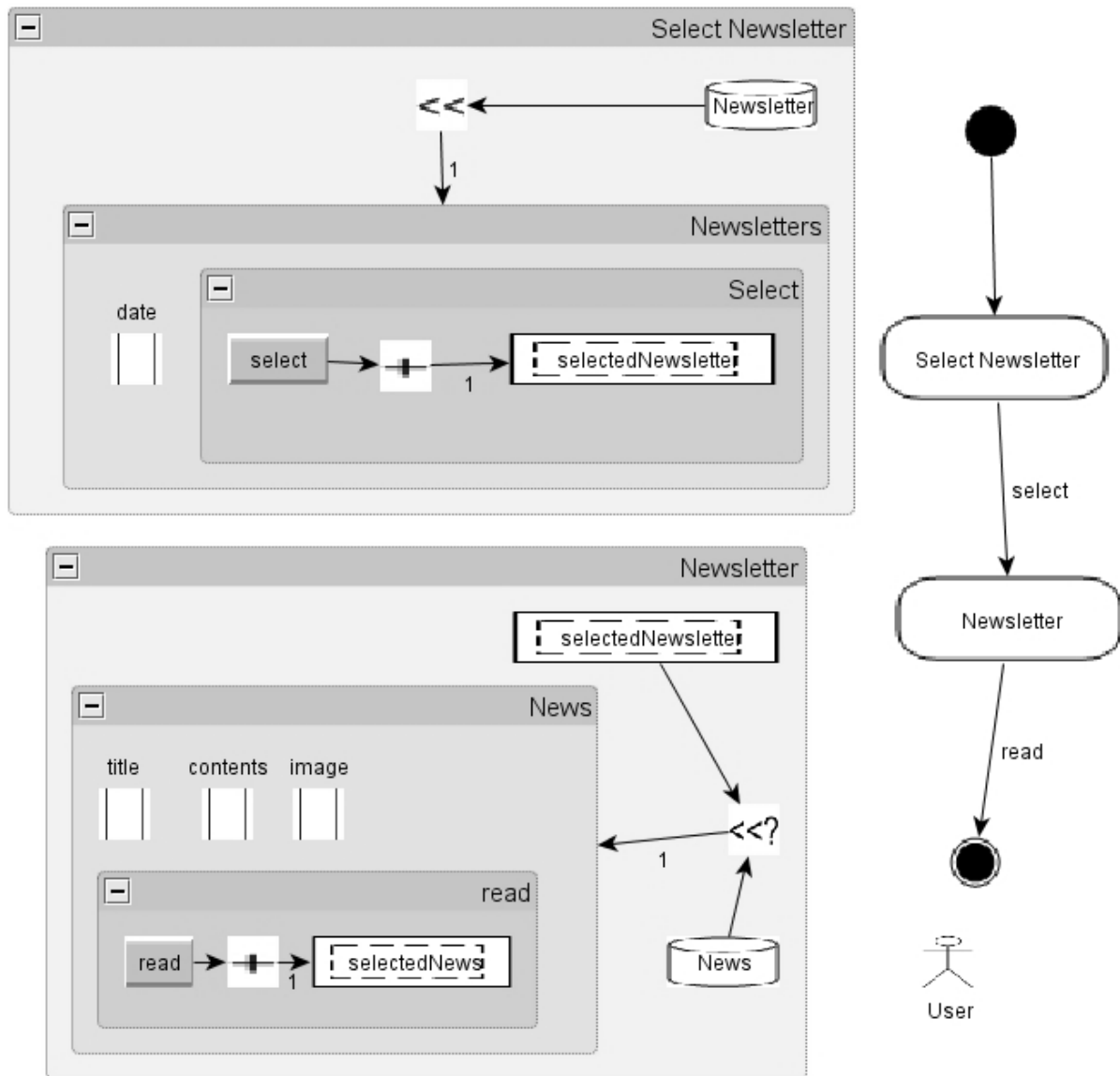


Figure 148 The *pim* of *Open Newsletter* use case

C.1.23 Pay Online

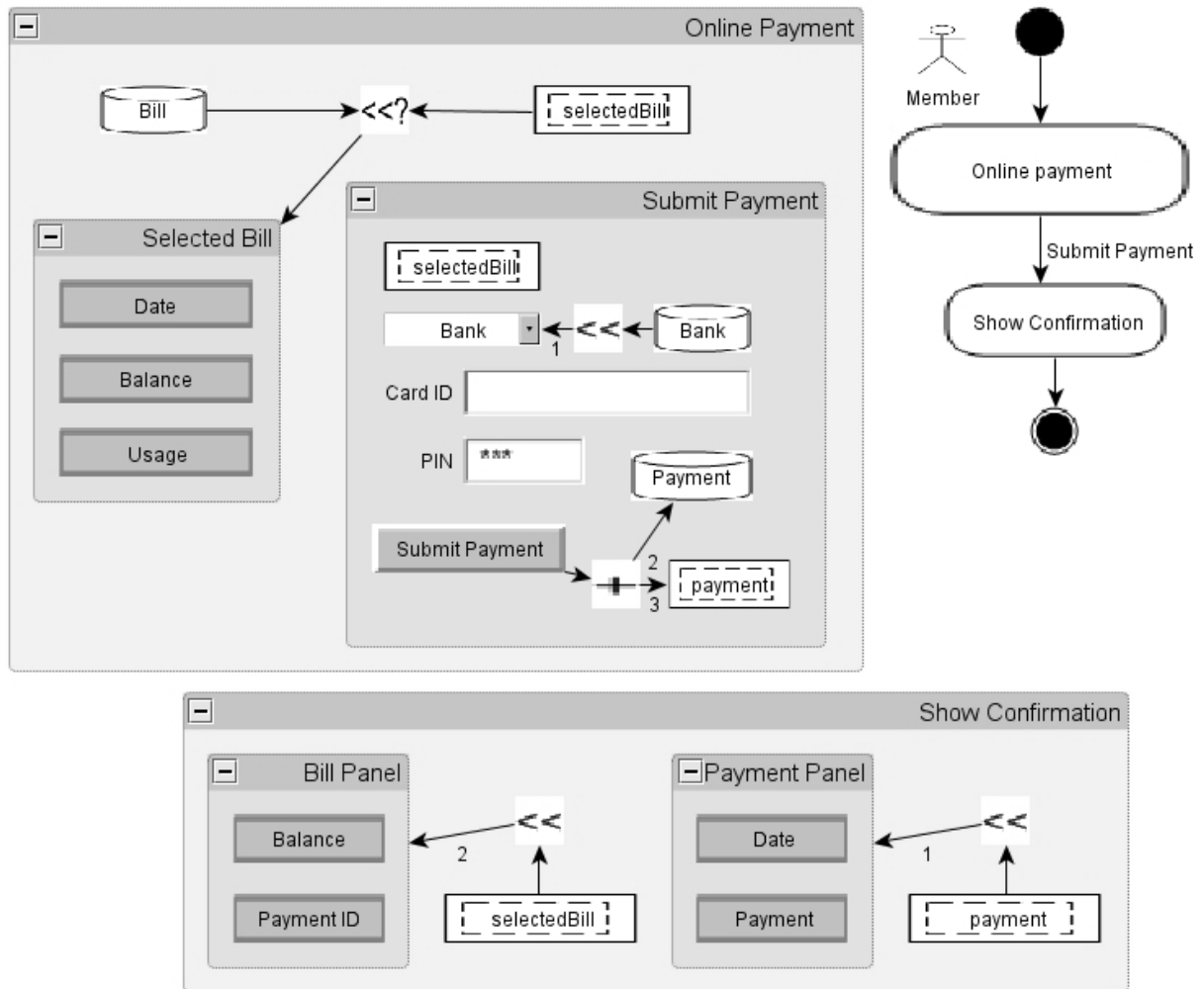


Figure 149 The *pim* of *Pay Online* use case

C.1.24 Print Bills

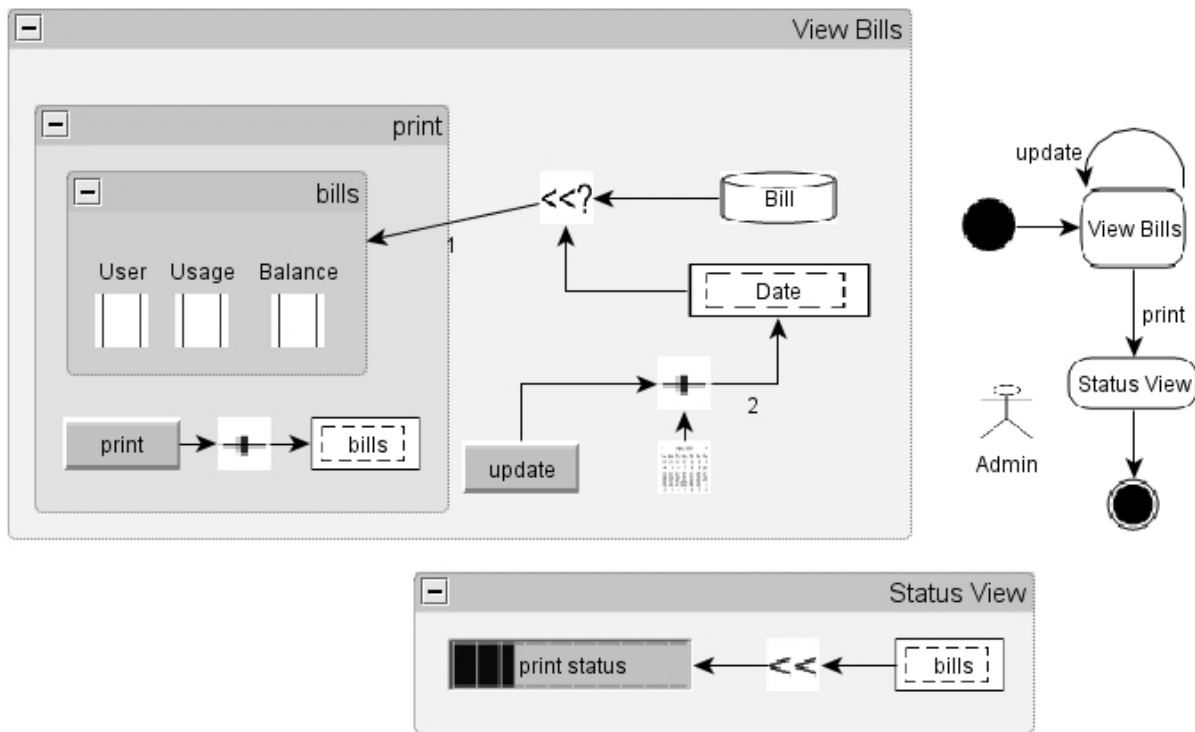


Figure 150 The *pim* of *Print Bills* use case

C.1.25 Process Instalment Application

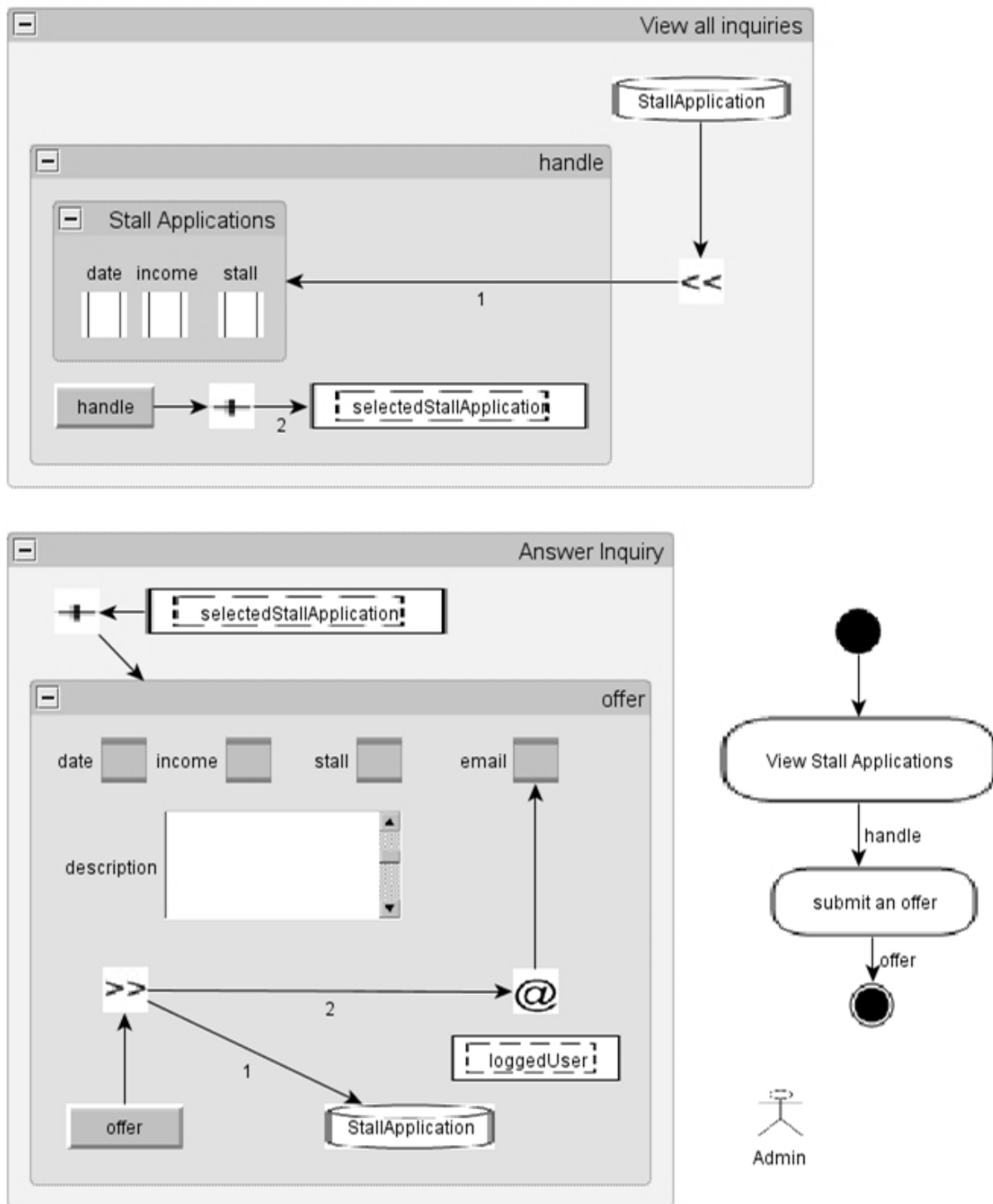


Figure 151 The *pim* of *Process Instalment Application* use case

C.1.26 Read News

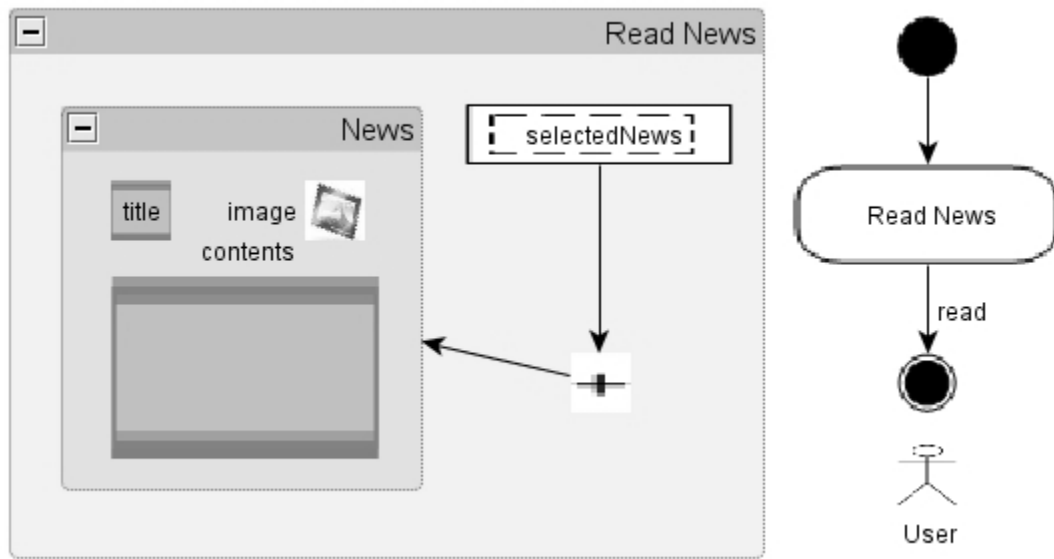


Figure 152 The *pim* of *Read News* use case

C.1.27 Remove Admin

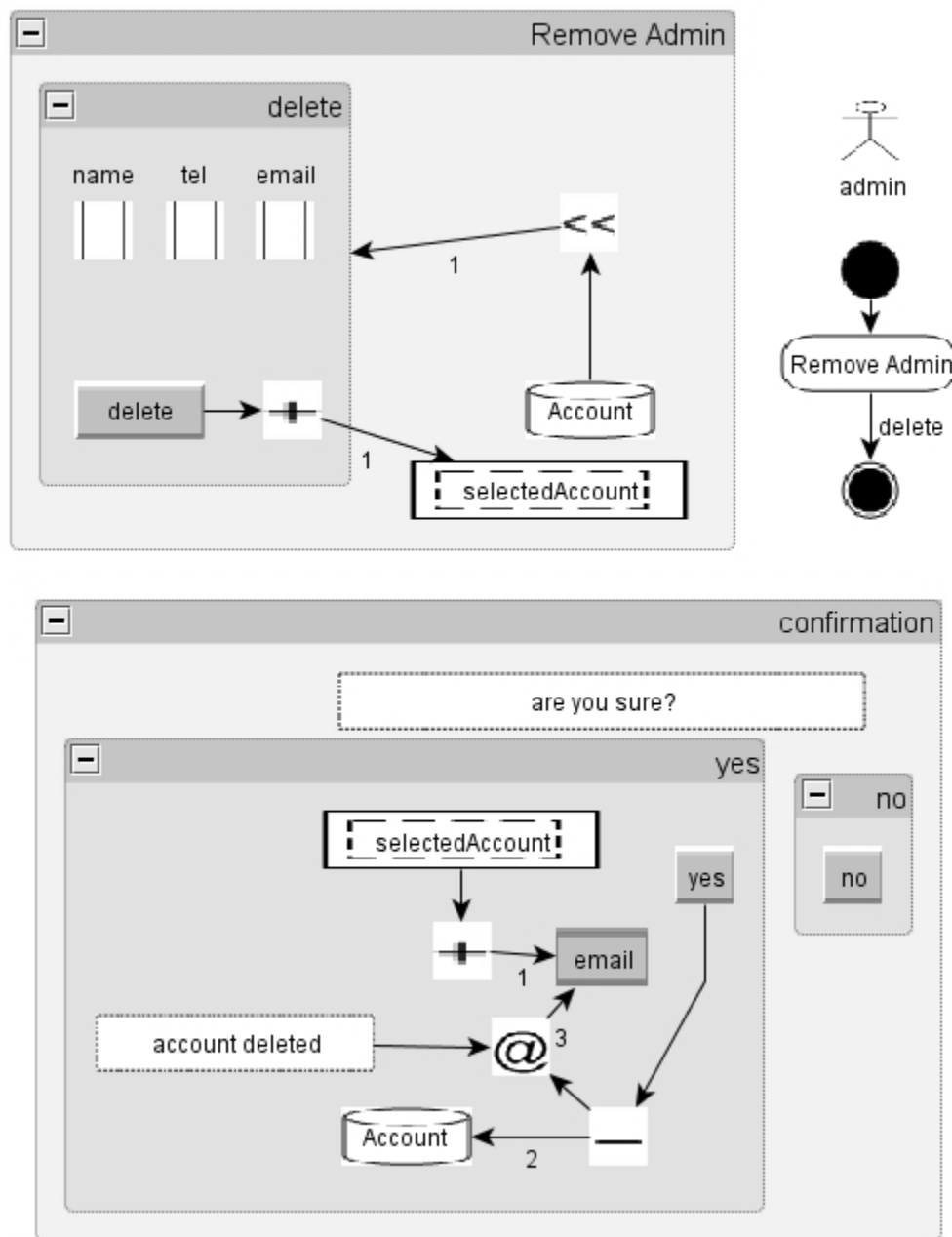


Figure 153 The *pim* of *Remove Admin* use case

C.1.28 Remove Rule

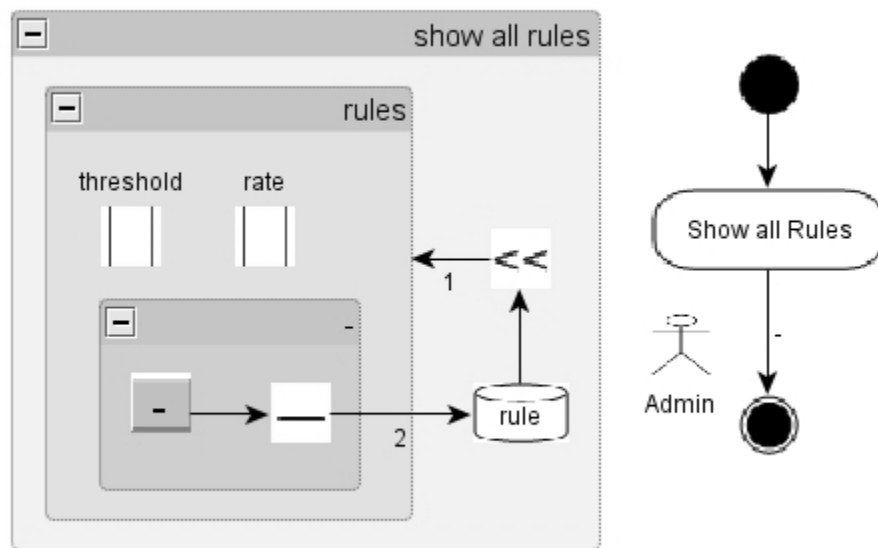


Figure 154 The *pim* of the *Remove Rule* pim

C.1.29 Remove Paper/EBill

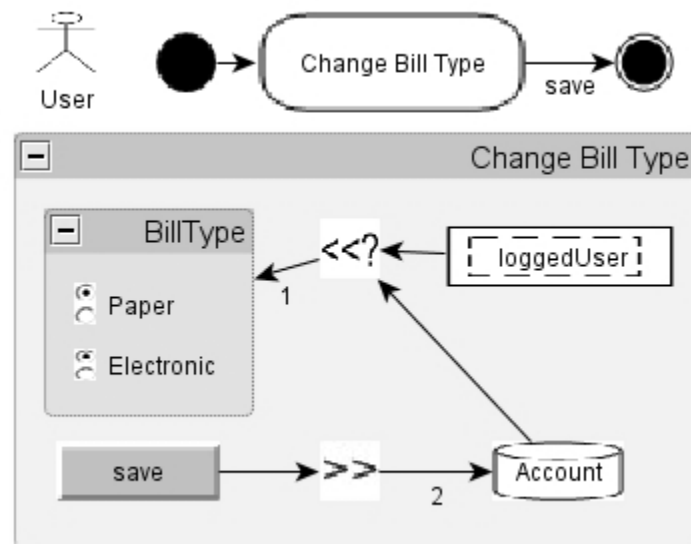


Figure 155 The *pim* of *Request Paper/EBill* use case

C.1.30 Apply for Instalments

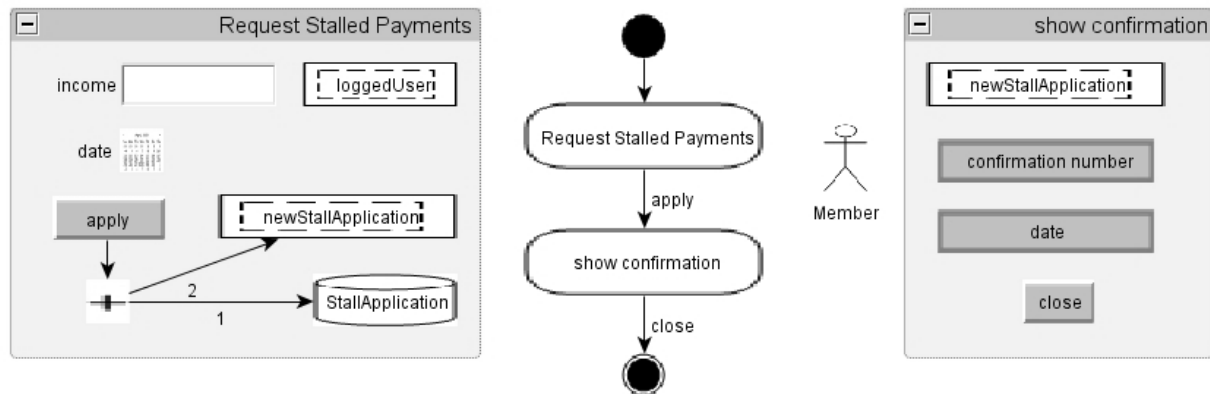


Figure 156 The *pim* of *Apply for Instalments* use case

C.1.31 Request an Account

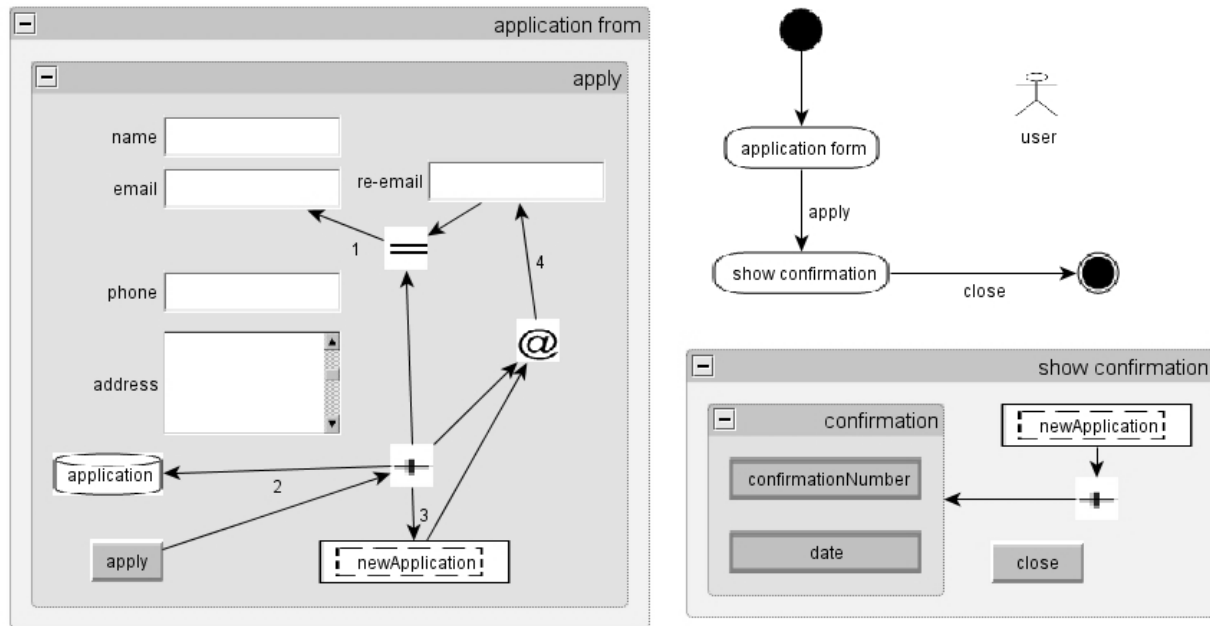


Figure 157 The *pim* of *Request an Account* use case

C.1.32 Request an Appointment

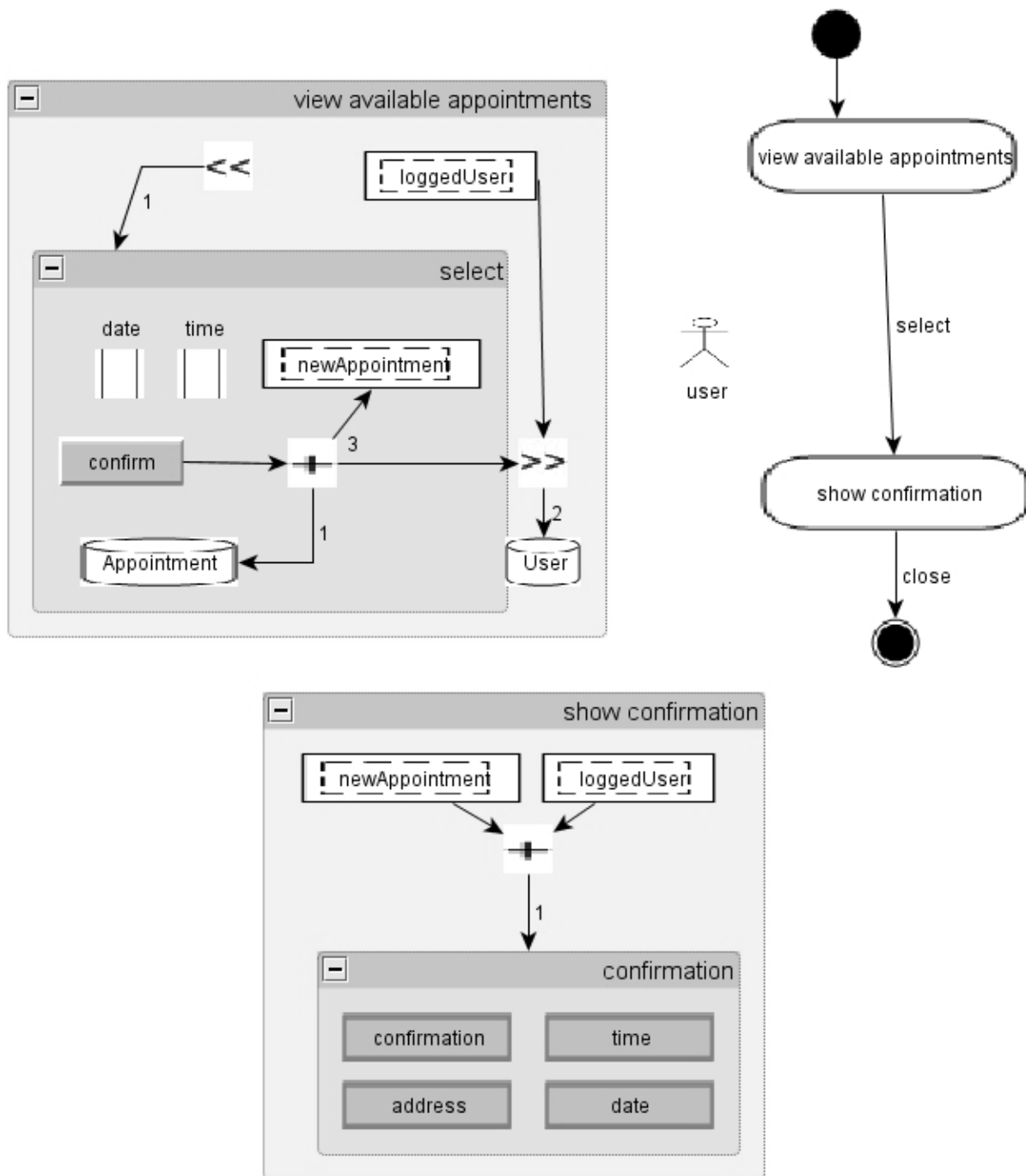


Figure 158 The *pim* of *Request an Appointment* use case

C.1.33 Search Tutorial

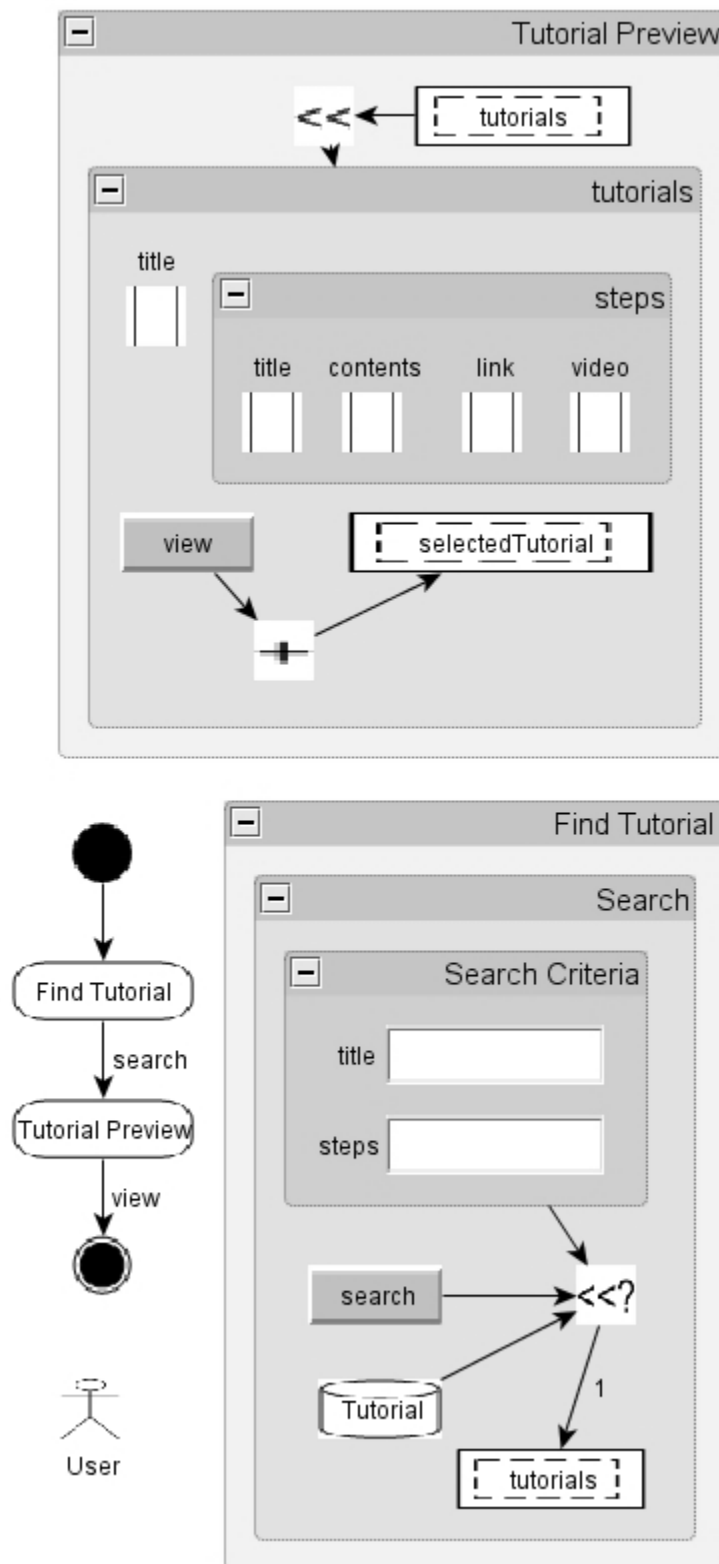


Figure 159 The *pim* of *Search Tutorial* use case

C.1.34 View Bill

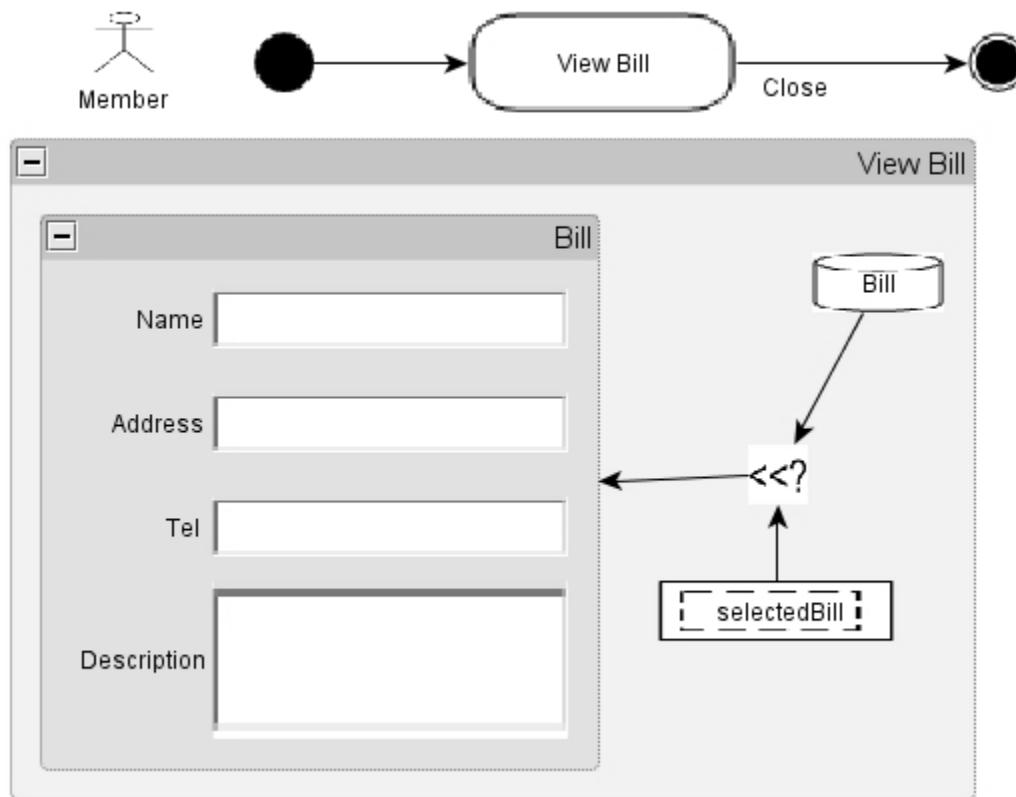


Figure 160 The *pim* of *View Bill* use case

C.1.35 View Billing Formula

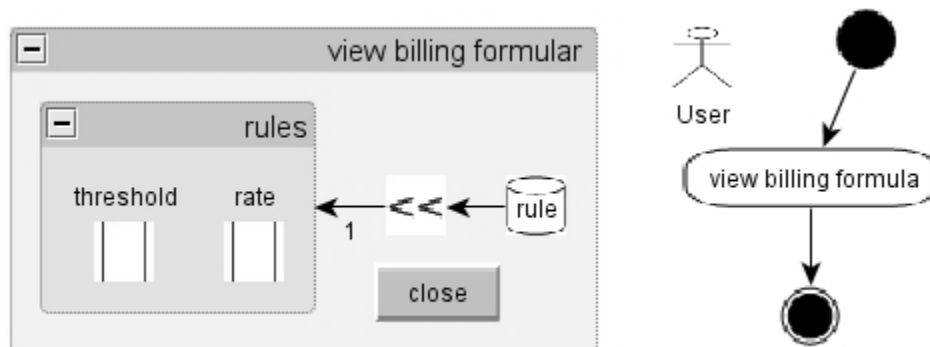


Figure 161 The *pim* of *View Billing Formula* use case

C.1.36 View Bills

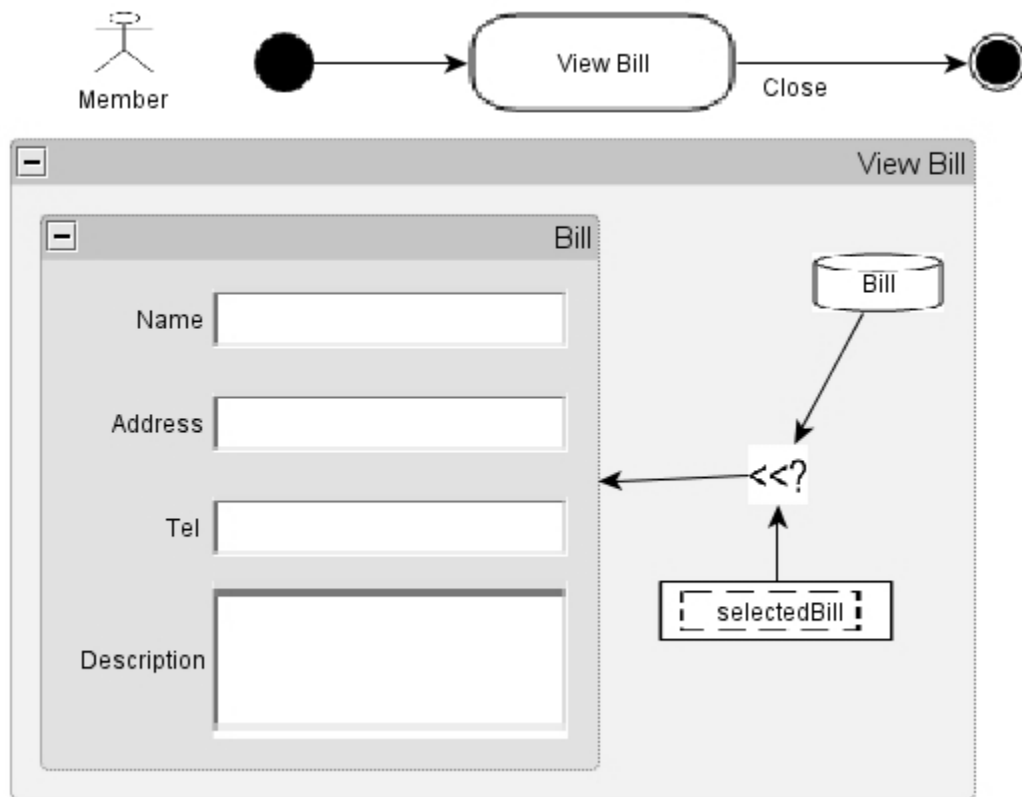


Figure 162 The *pim* of *View Bills* use case

C.1.37 View Tutorial

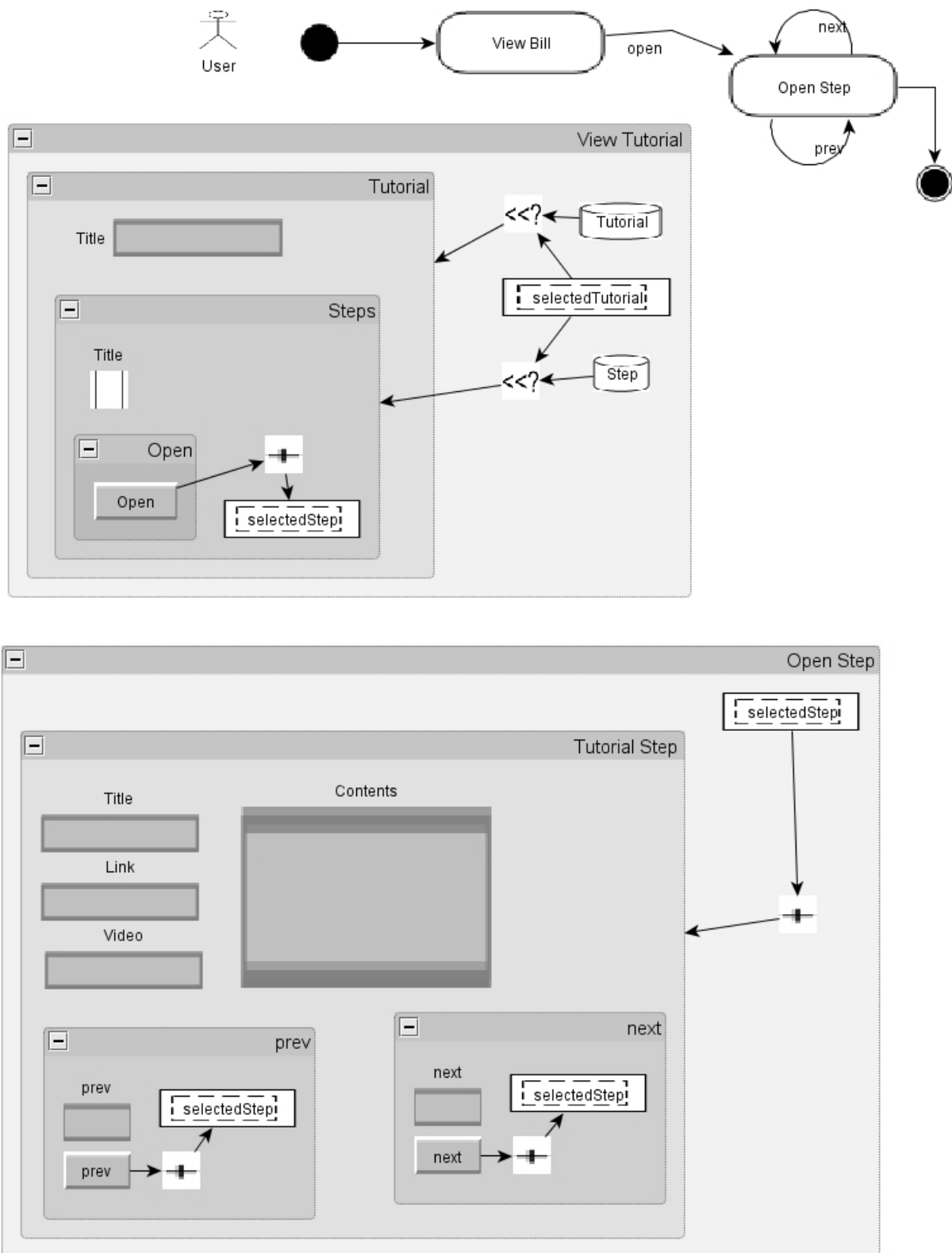


Figure 163 The *pim* of *View Tutorial* use case

C.2 AMS apsm

C.2.1 Add Admin Use Case

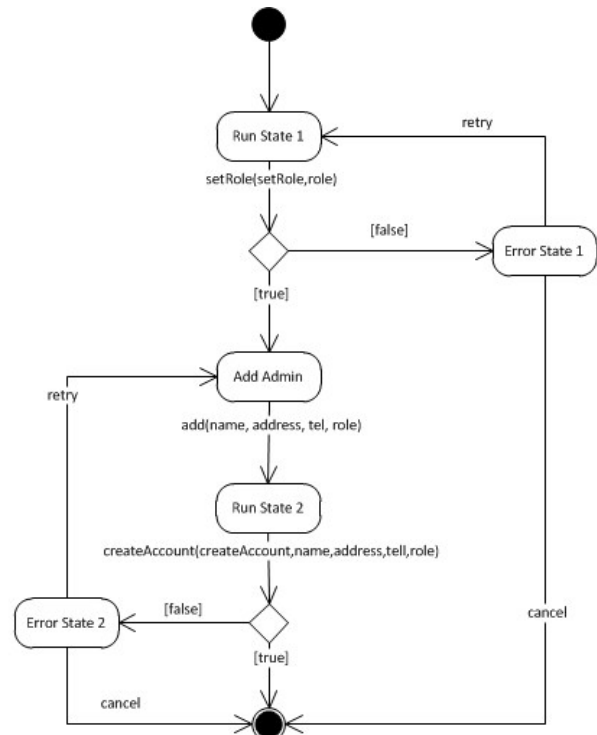
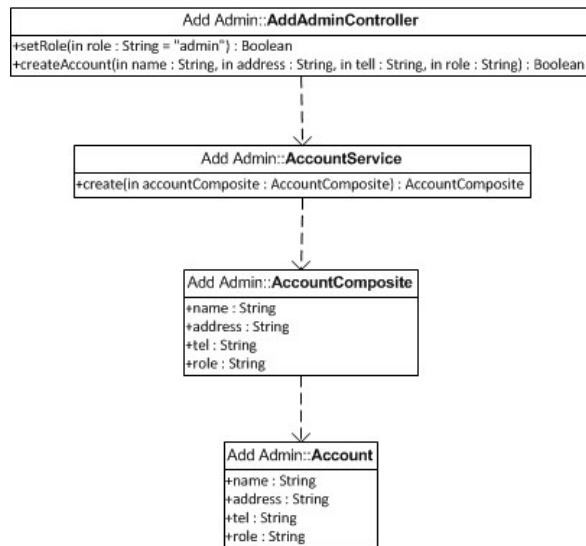


Figure 164 The *apsm* of the *Add Admin* use case

C.2.2 Add New Bank

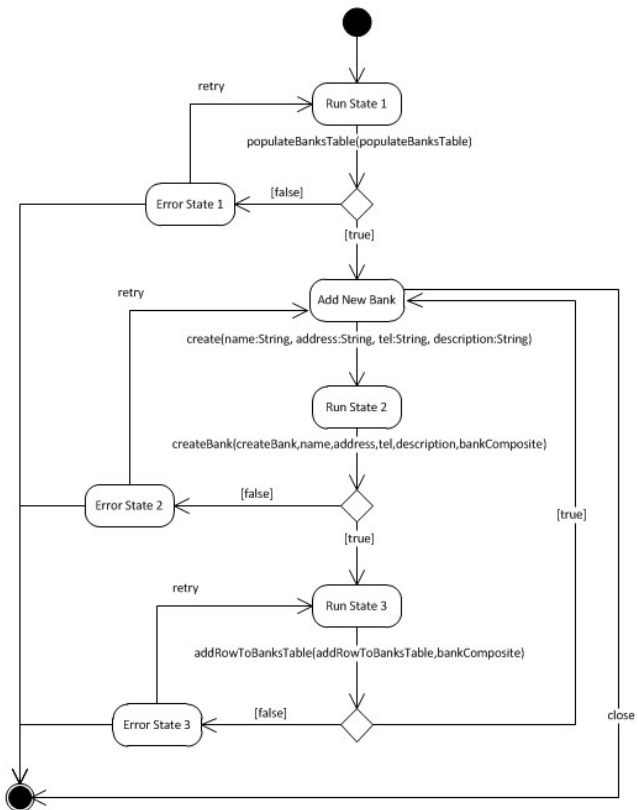
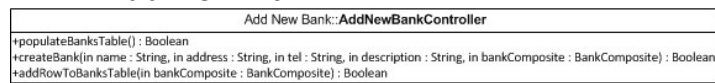


Figure 165 The *apsm* of the *Add New Bank* use case

C.2.3 Add News

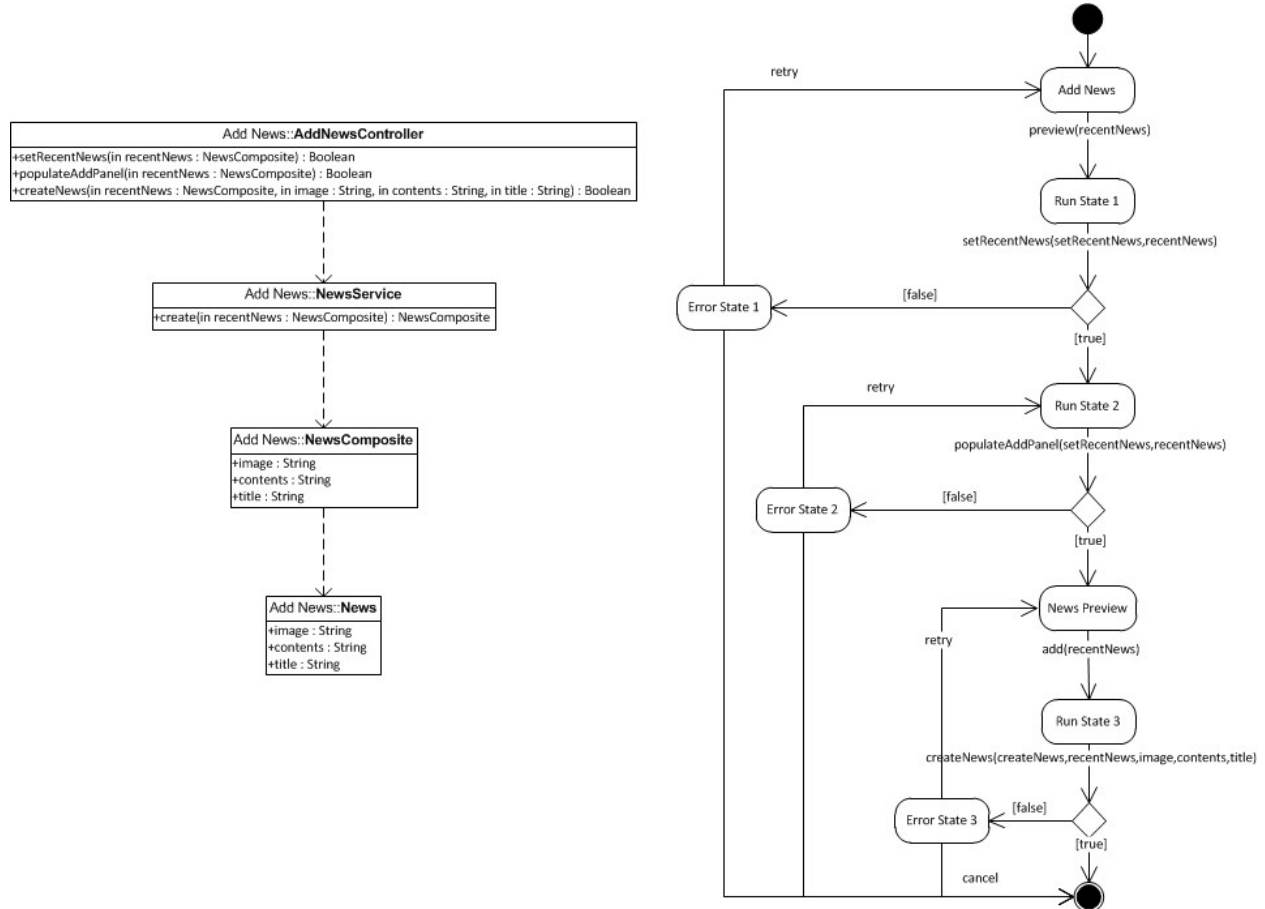


Figure 166 The *apsm* of the *Add News* use case

C.2.4 Add Tutorial

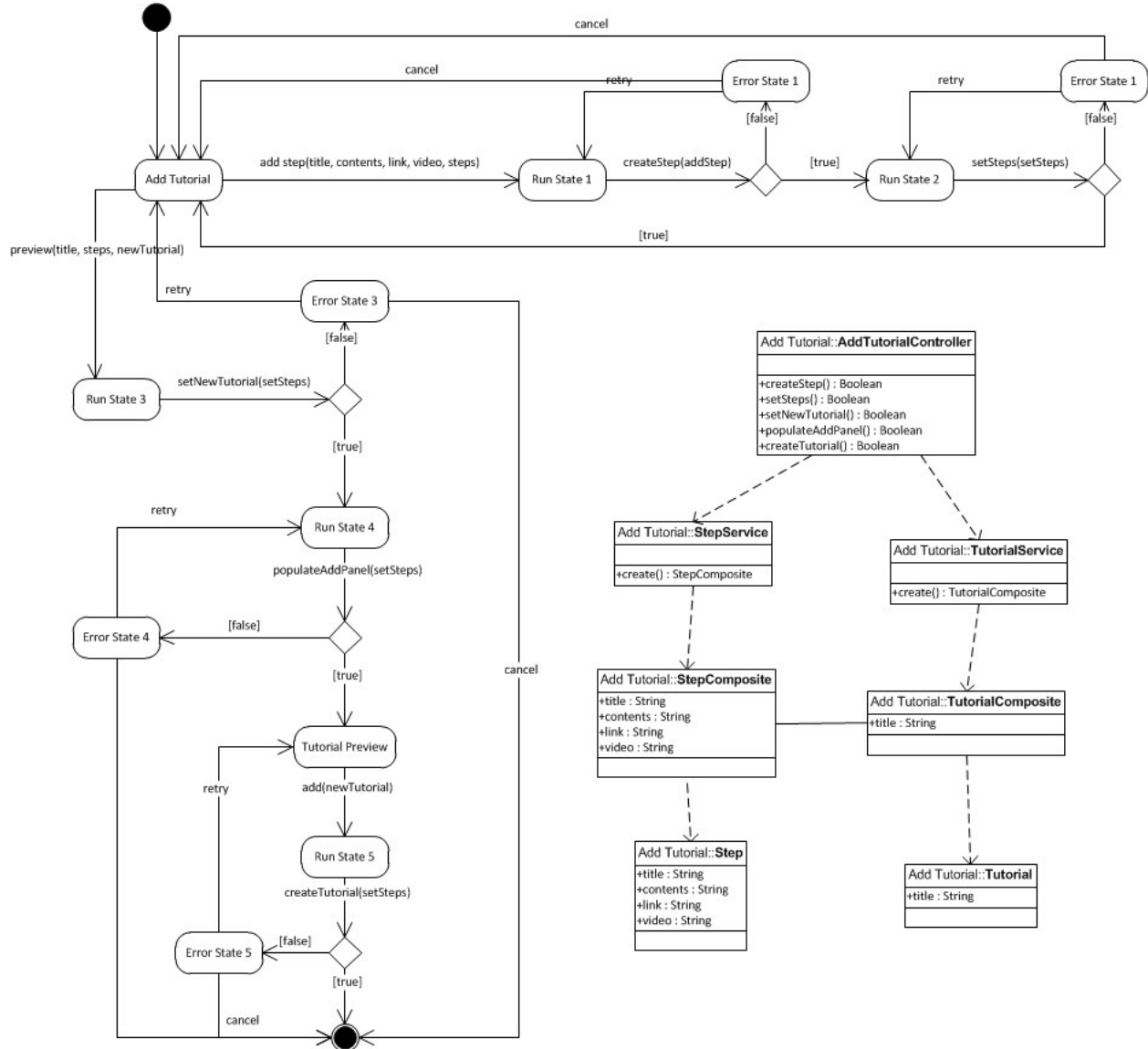


Figure 167 The *apsm* of the *Add Tutorial* use case

C.2.5 Add User

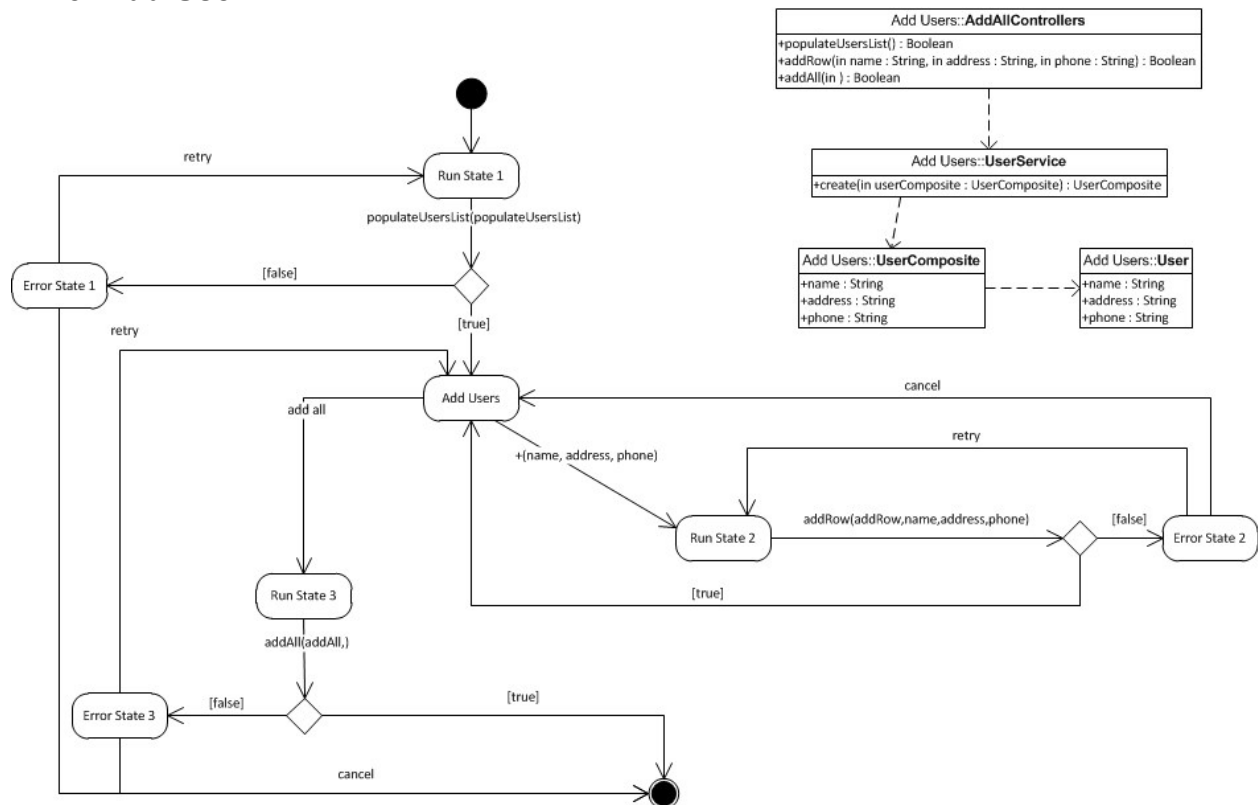


Figure 168 The apsm of the Add User use case

C.2.6 Add Billing Rule

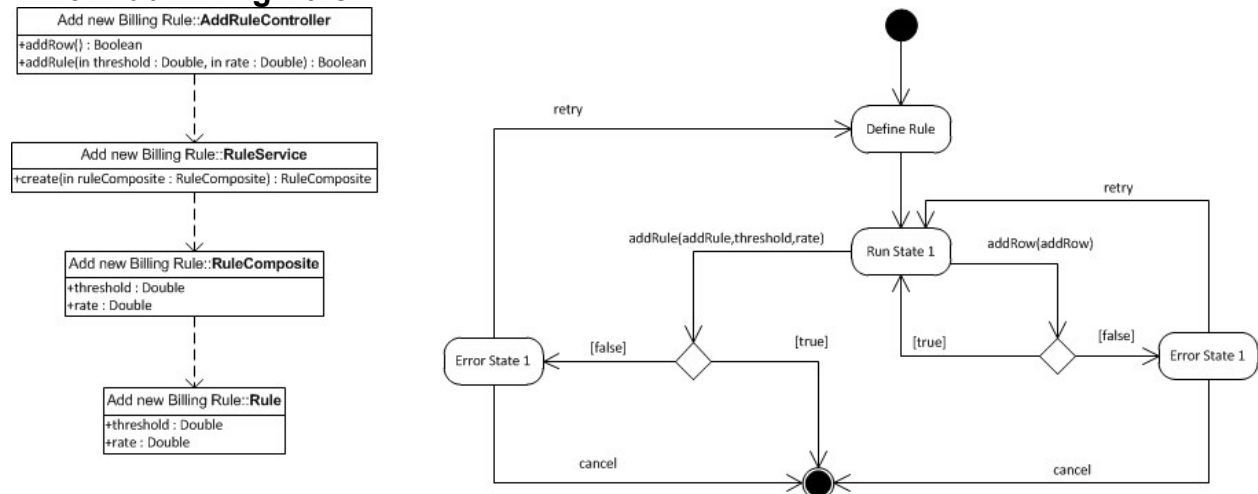


Figure 169 The apsm of the Add Billing Rule use case

C.2.7 Address an Inquiry

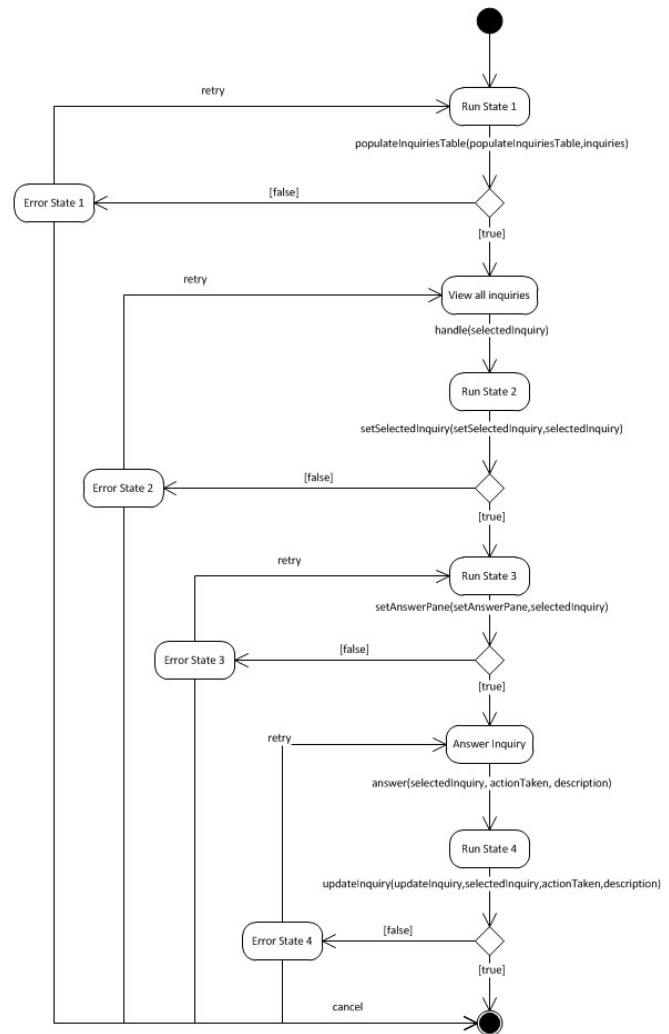
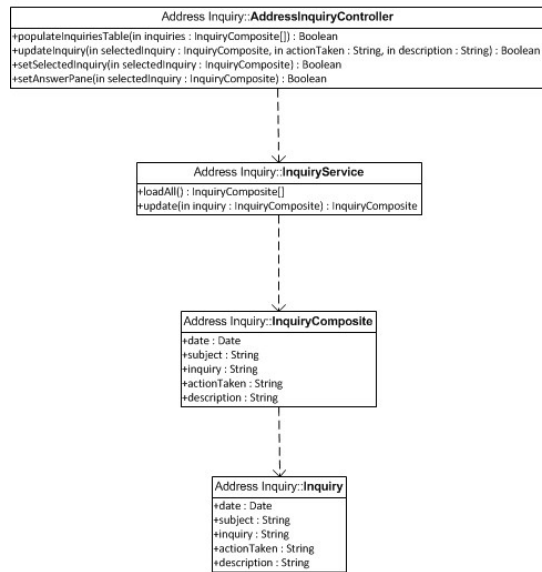


Figure 170 The *apsm* of the *Address an Inquiry* use case

C.2.8 Apply Penalty

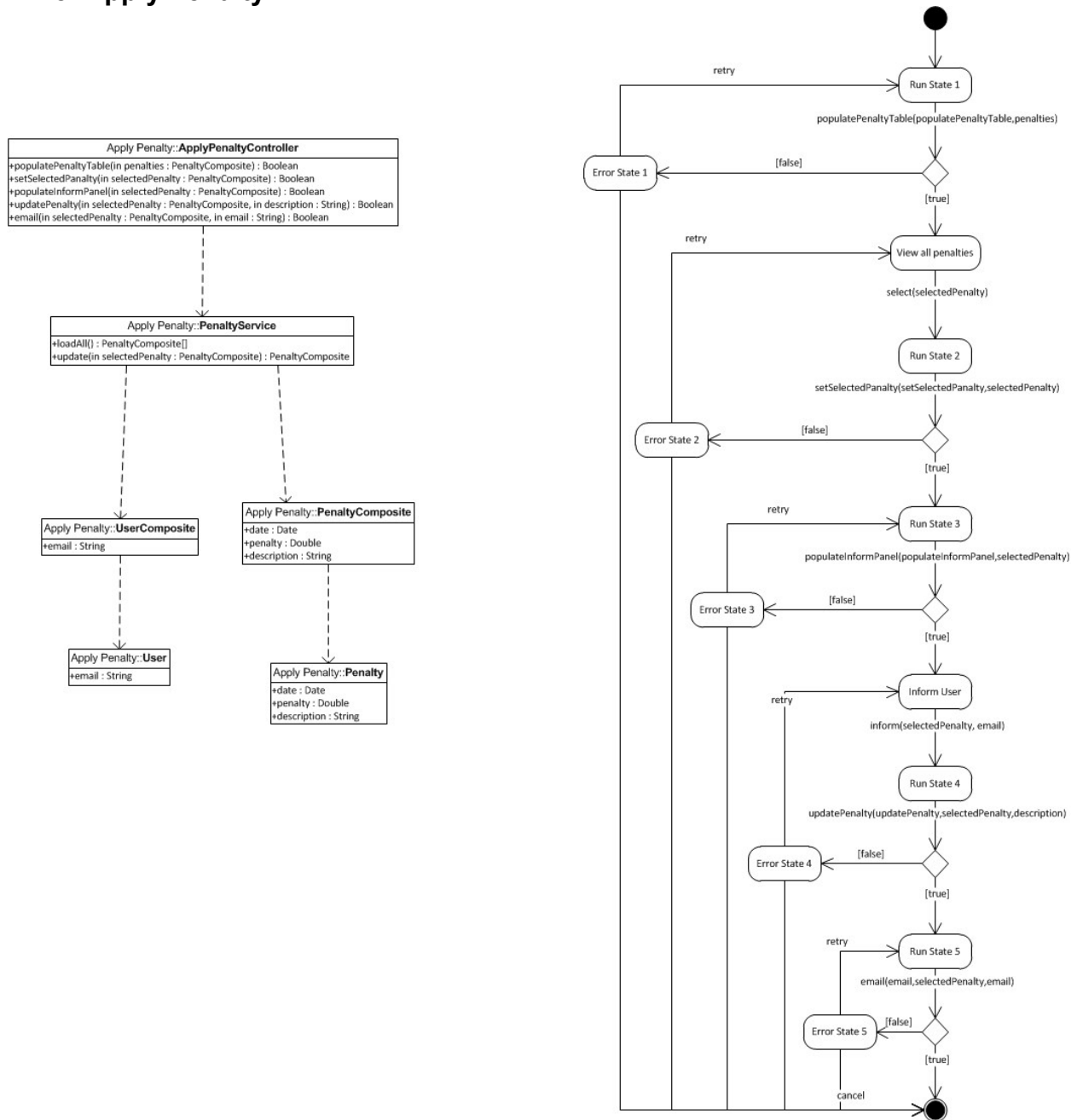


Figure 171 The *apsm* of the *Apply Penalty* use case

C.2.9 Assign Account

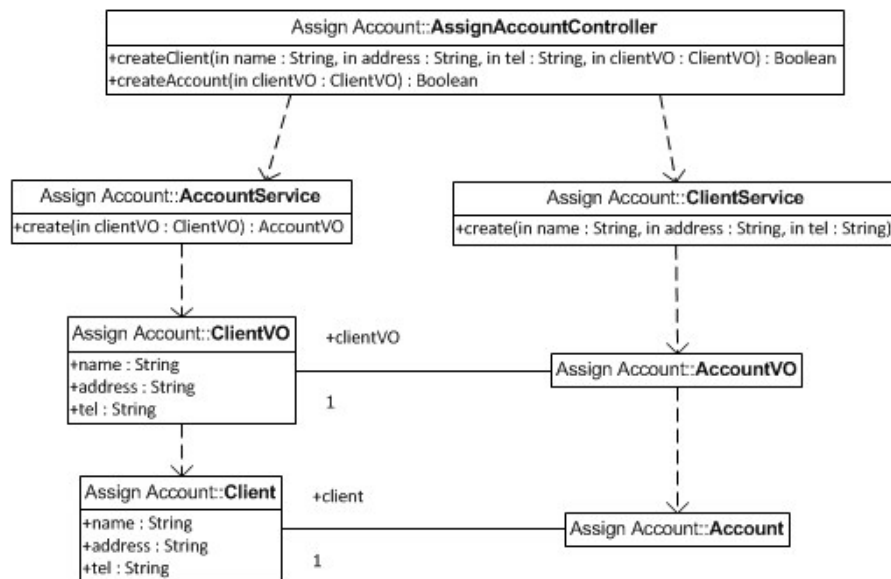
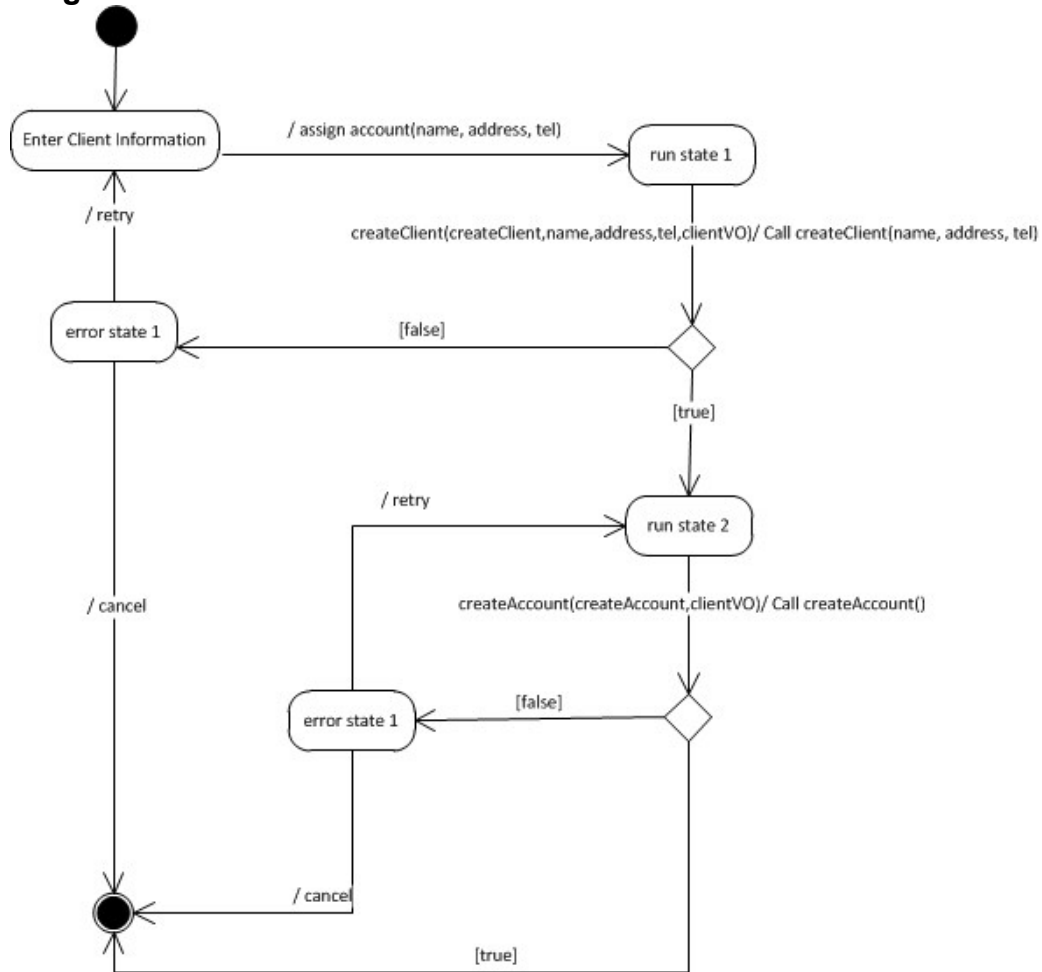


Figure 172 The *apsm* of the *Assign Account* use case

C.2.10 Cancel Instalment

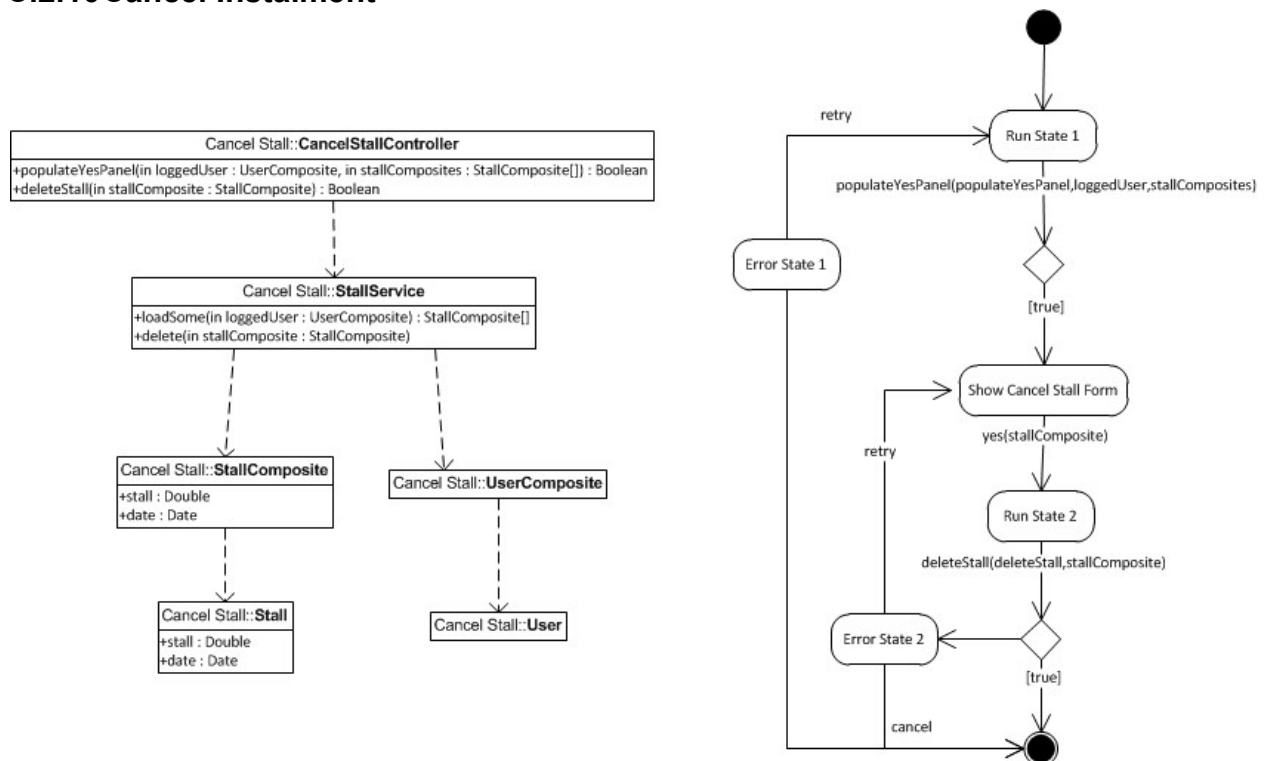


Figure 173 The *apsm* of the *Cancel Instalment* use case

C.2.11 Confirm Account Request

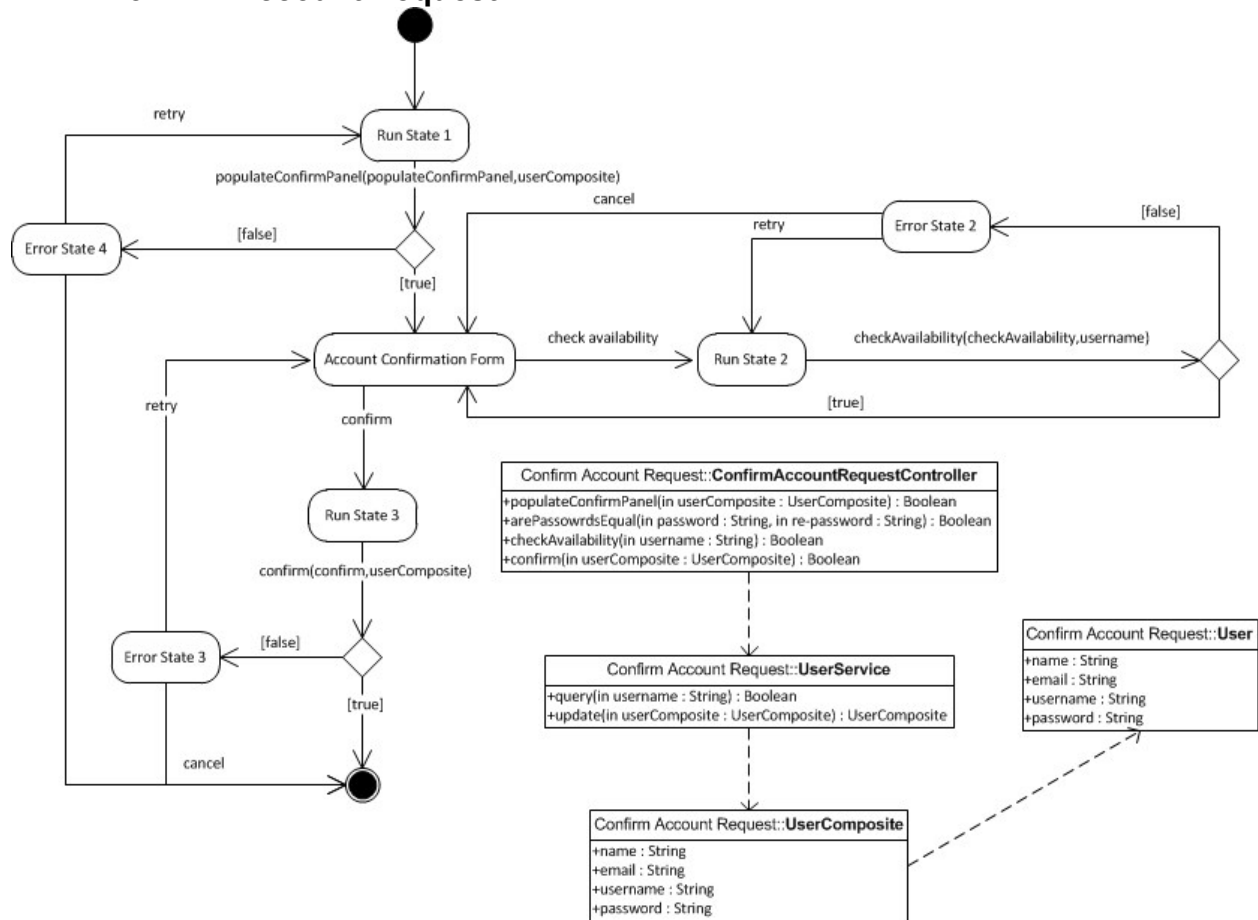


Figure 174 The *apsm* of the *Confirm Account Request* use case

C.2.12 Confirm Instalment Offer

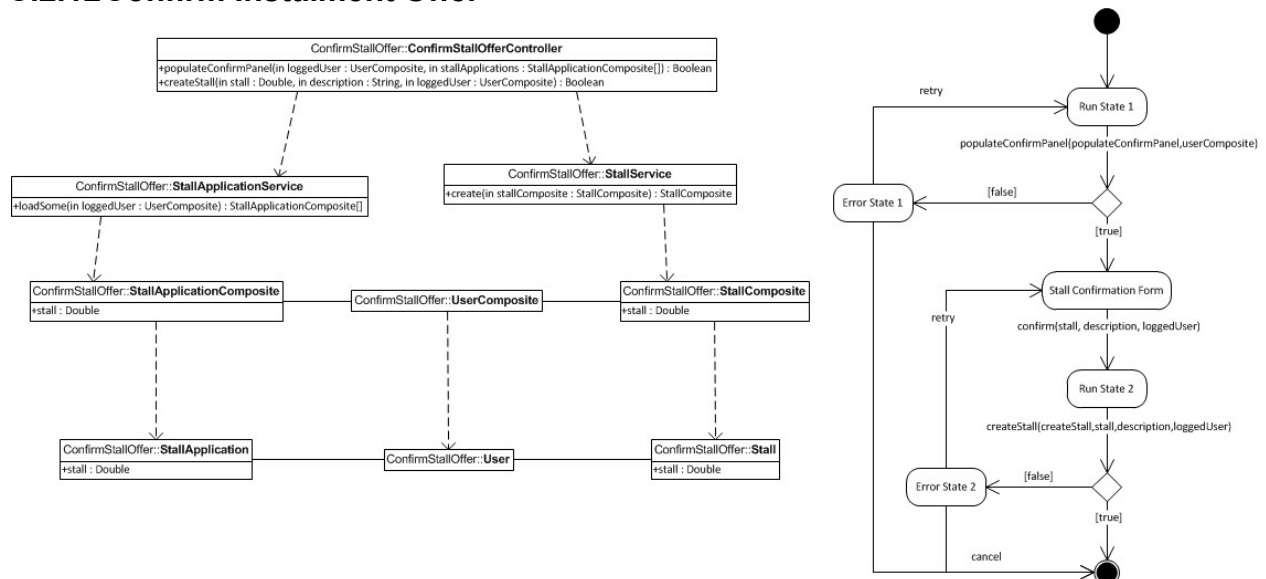


Figure 175 The *apsm* of the *Confirm Instalment Offer* use case

C.2.13 Create Newsletter

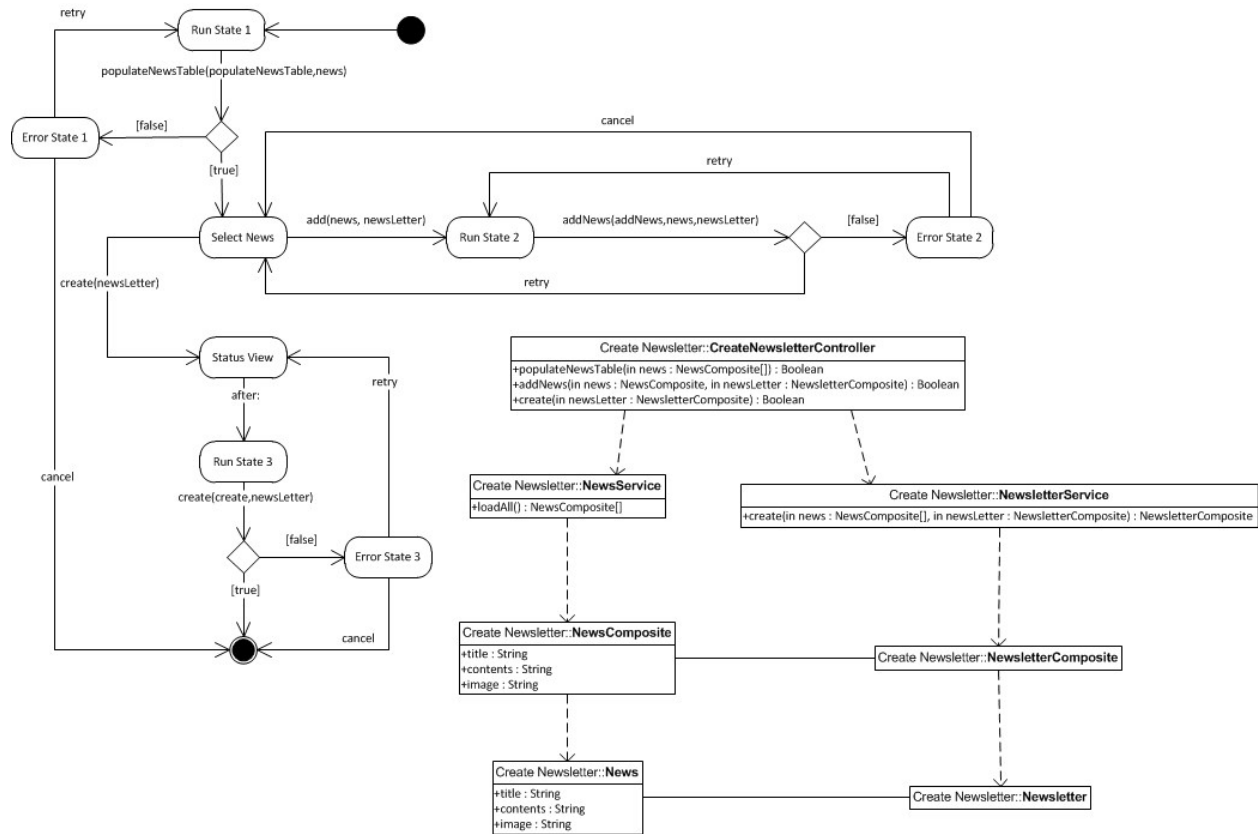


Figure 176 The apsm of the *Create Newsletter* use case

C.2.14 Edit Billing Rule

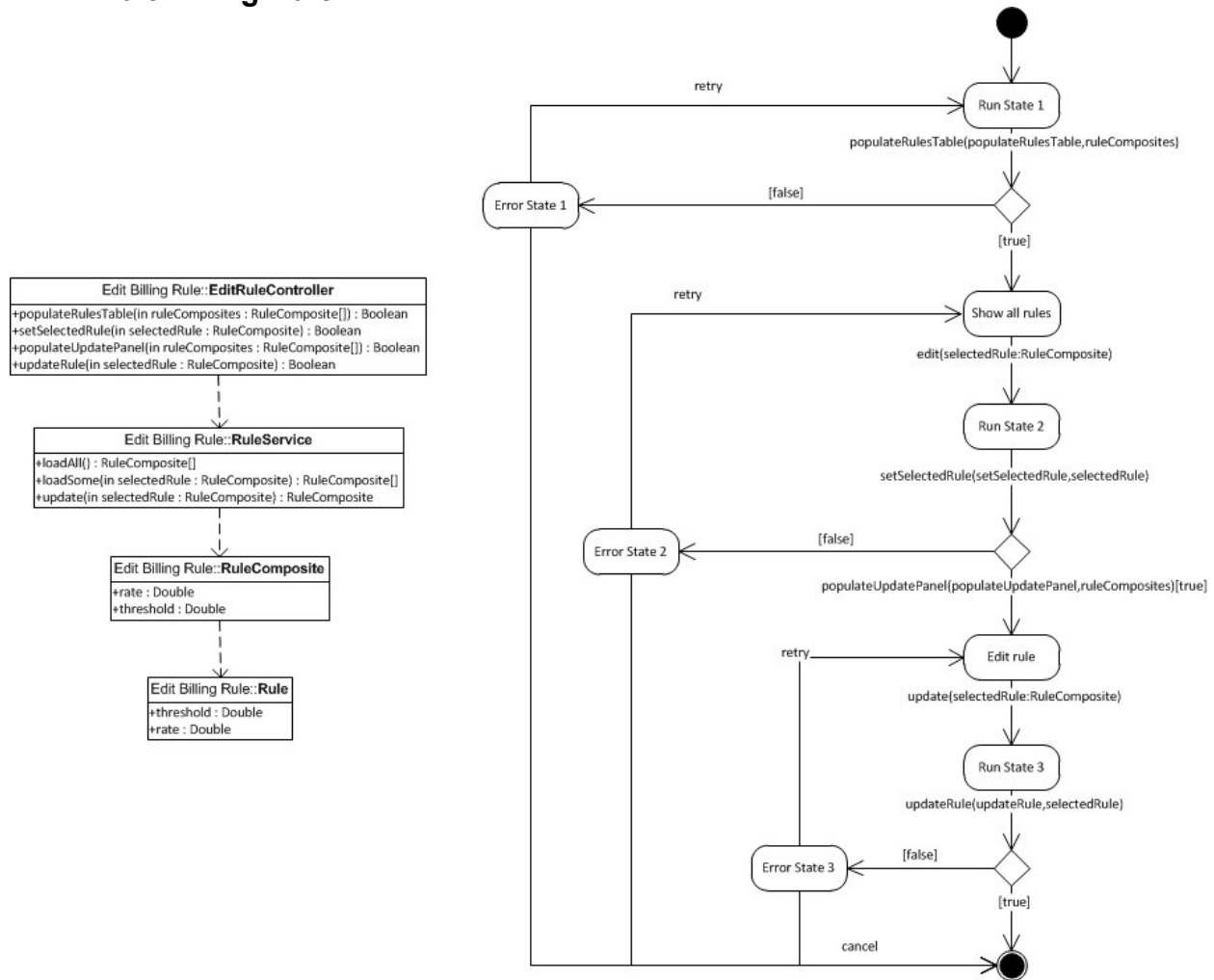


Figure 177 The *apsm* of *Edit Rule* use case

C.2.15 Edit Personal Information

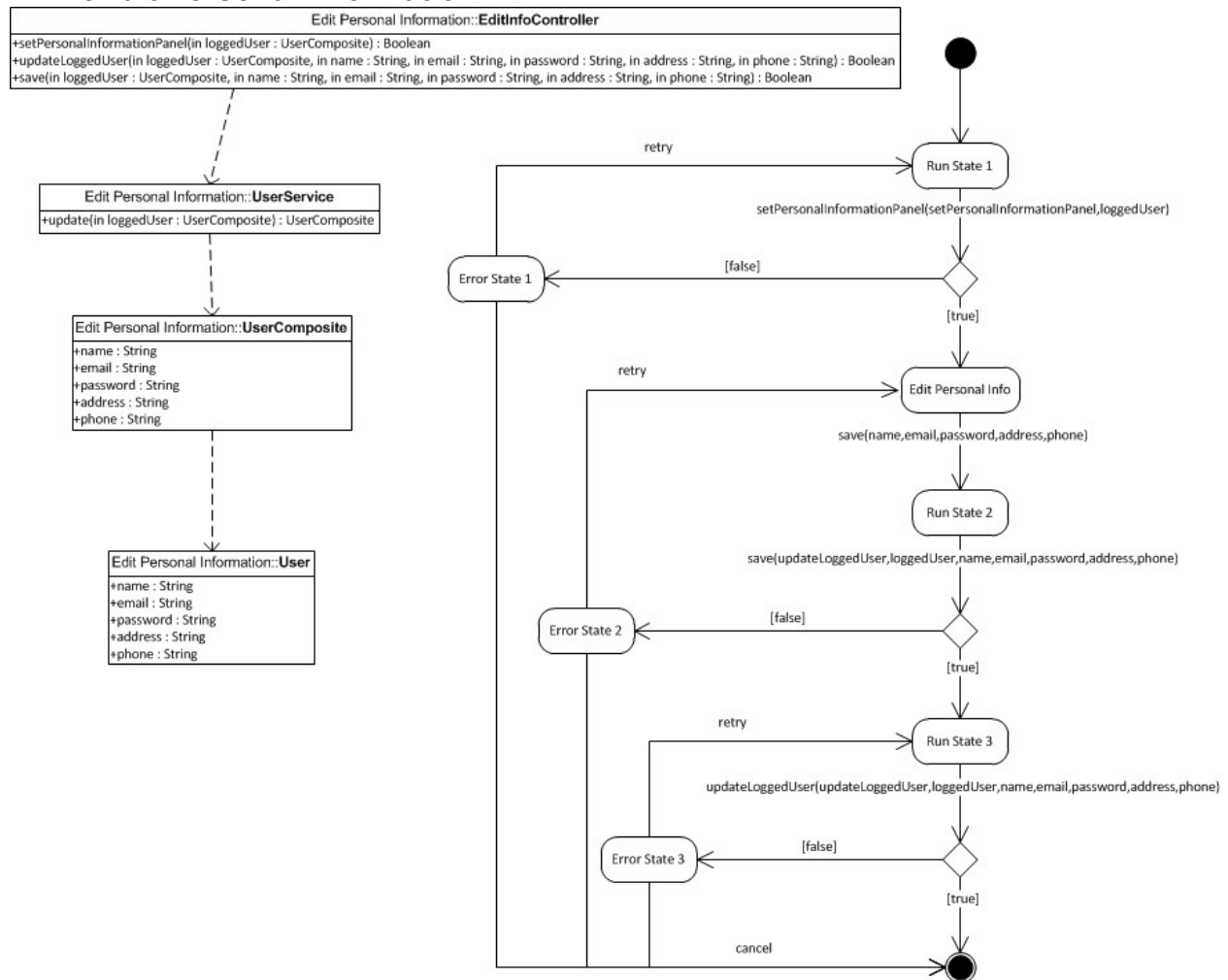


Figure 178 The *apsm* of *Edit Personal Information* use case

C.2.16 Handle Appointment

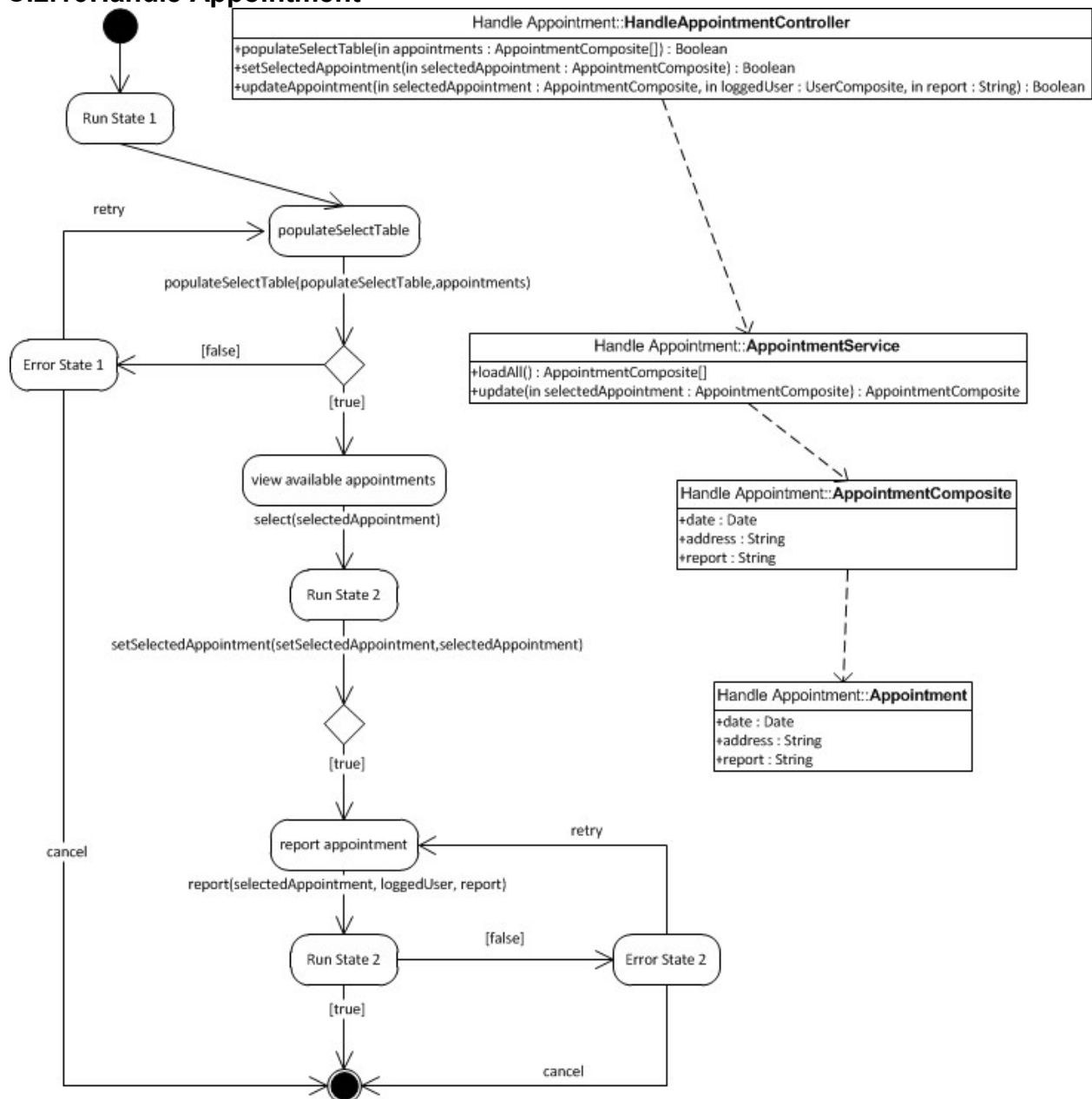


Figure 179 The *apsm* of *Handle Appointment* use case

C.2.17 Inquire a Bill

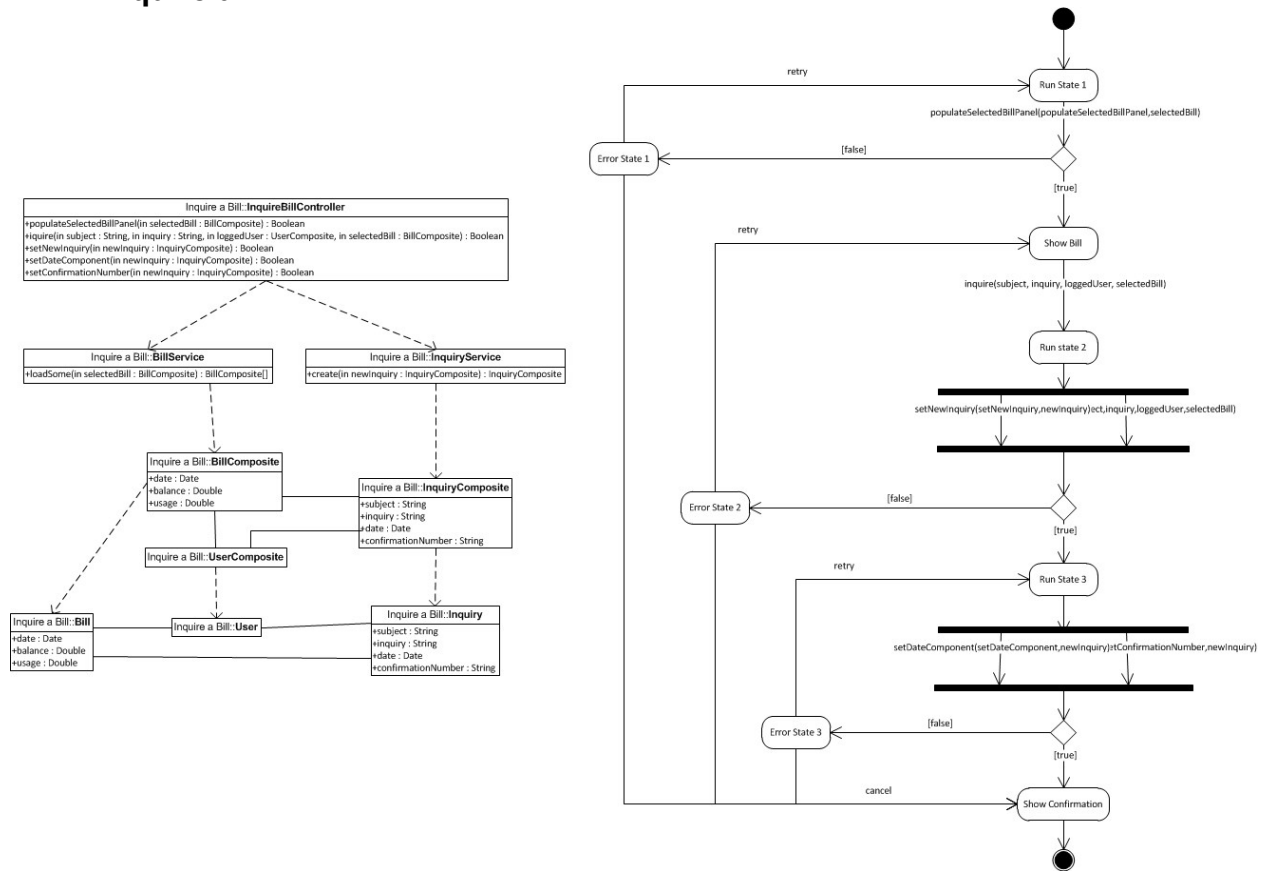


Figure 180 The *apsm* of *Inquire a Bill* use case

C.2.18 Login

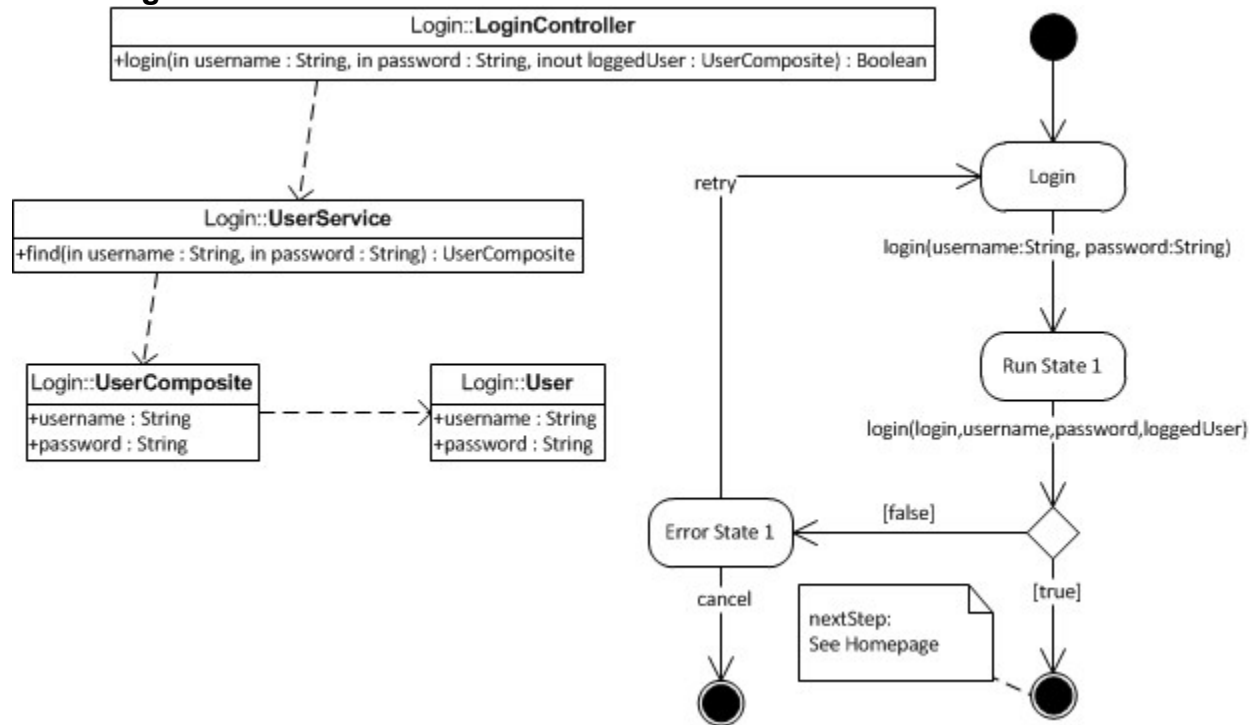


Figure 181 The *apsm* of Login use case

C.2.19 Subscribe to Newsletter

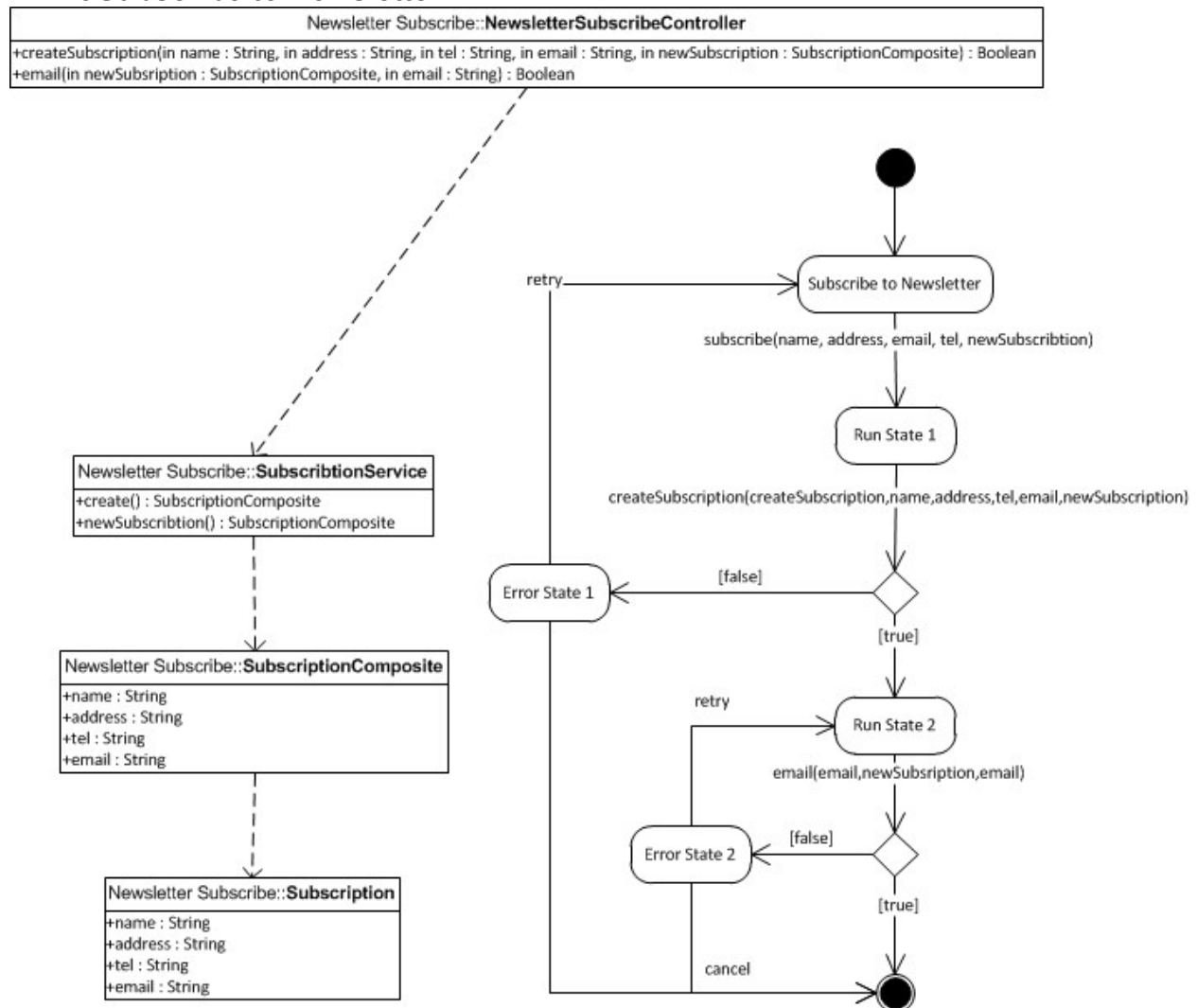


Figure 182 The *apsm* of *Subscribe to Newsletter* use case

C.2.20 Unsubscribe from Newsletter

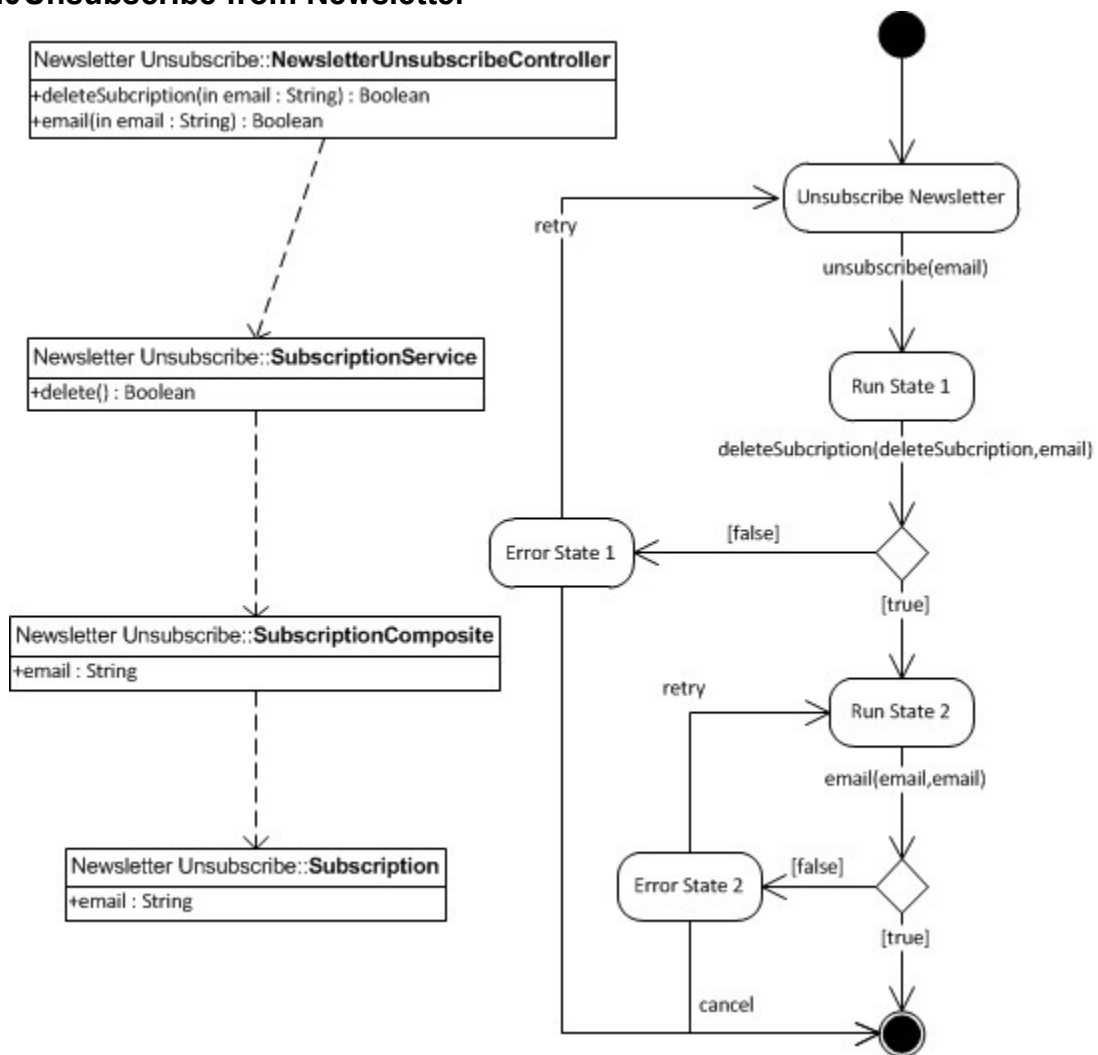


Figure 183 The *apsm* of *Unsubscribe from Newsletter* use case

C.2.21 Pay Online

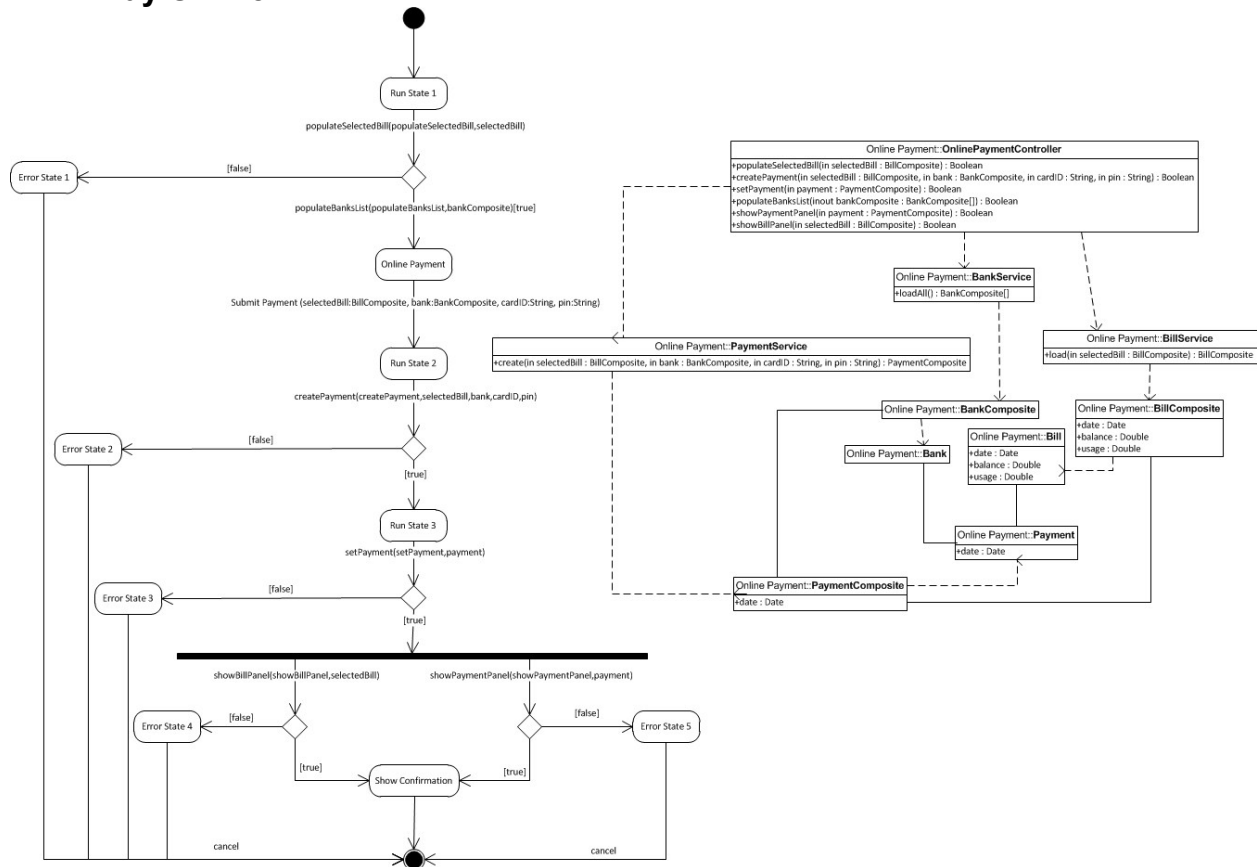


Figure 184 The apsm of Pay Online use case

C.2.22 Open Newsletter

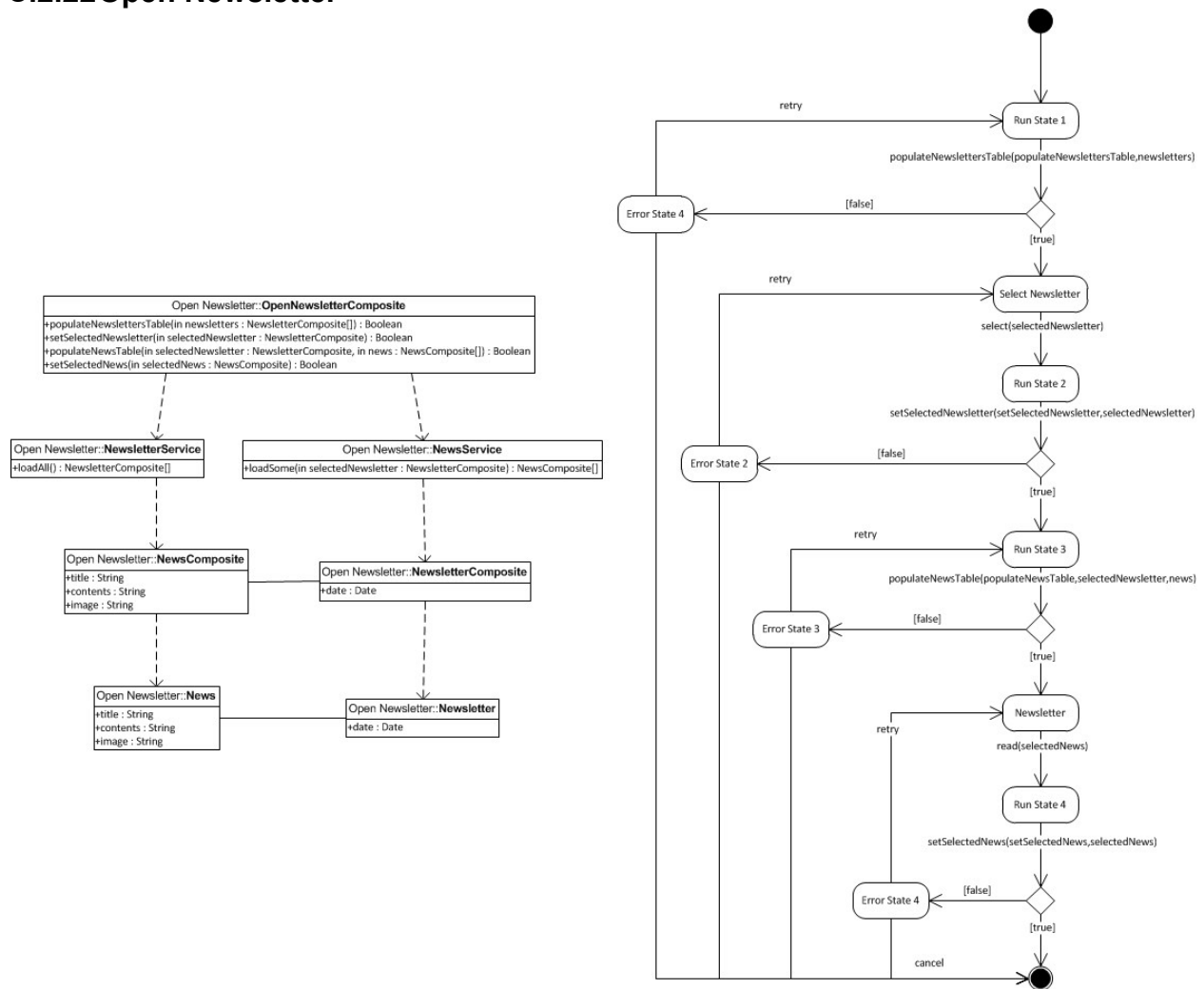


Figure 185 The *apsm* of *Open Newsletter* use case

C.2.23 Print Bills

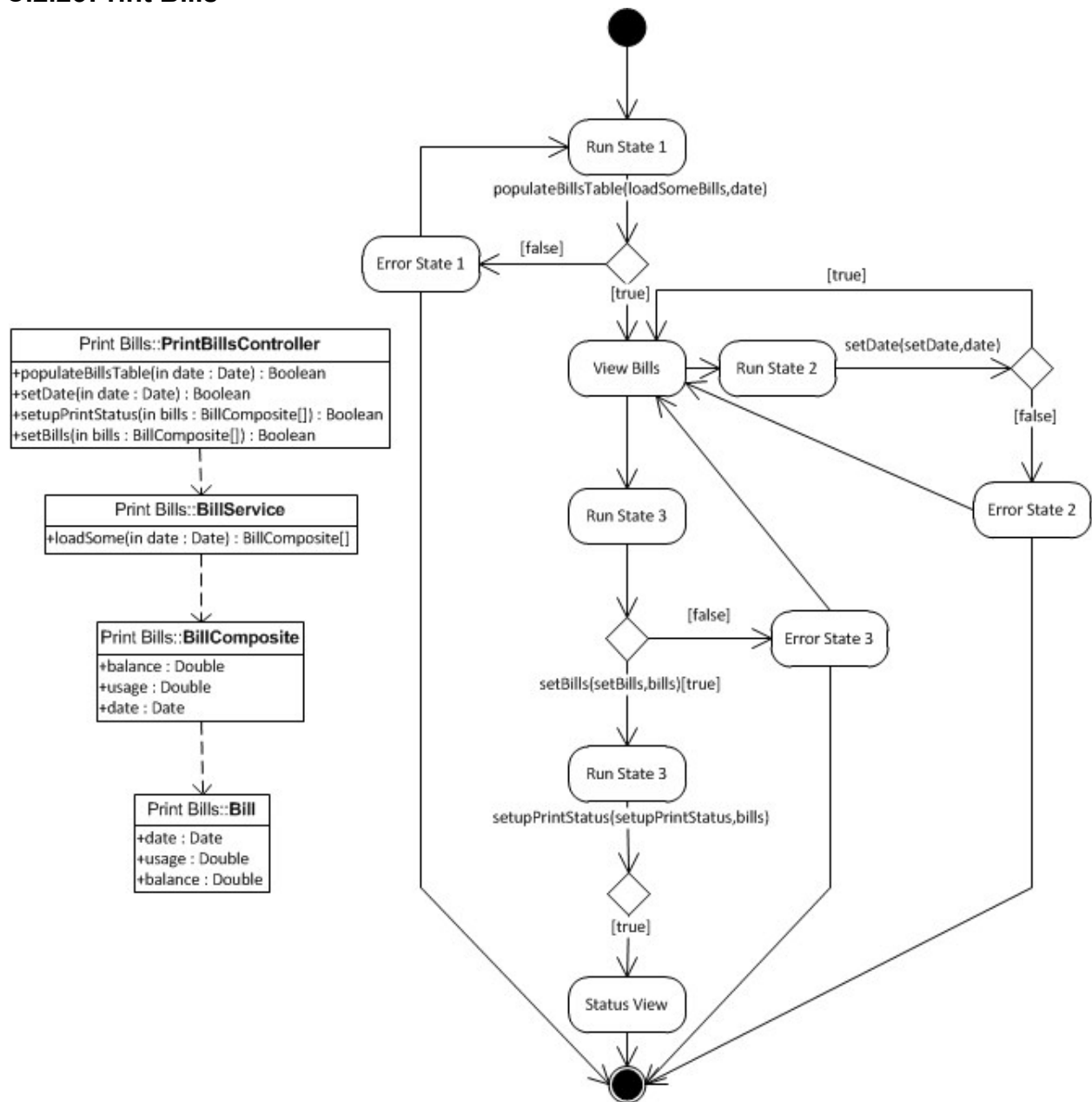


Figure 186 The *apsm* of *Print Bills* use case

C.2.24 Process Instalment Application

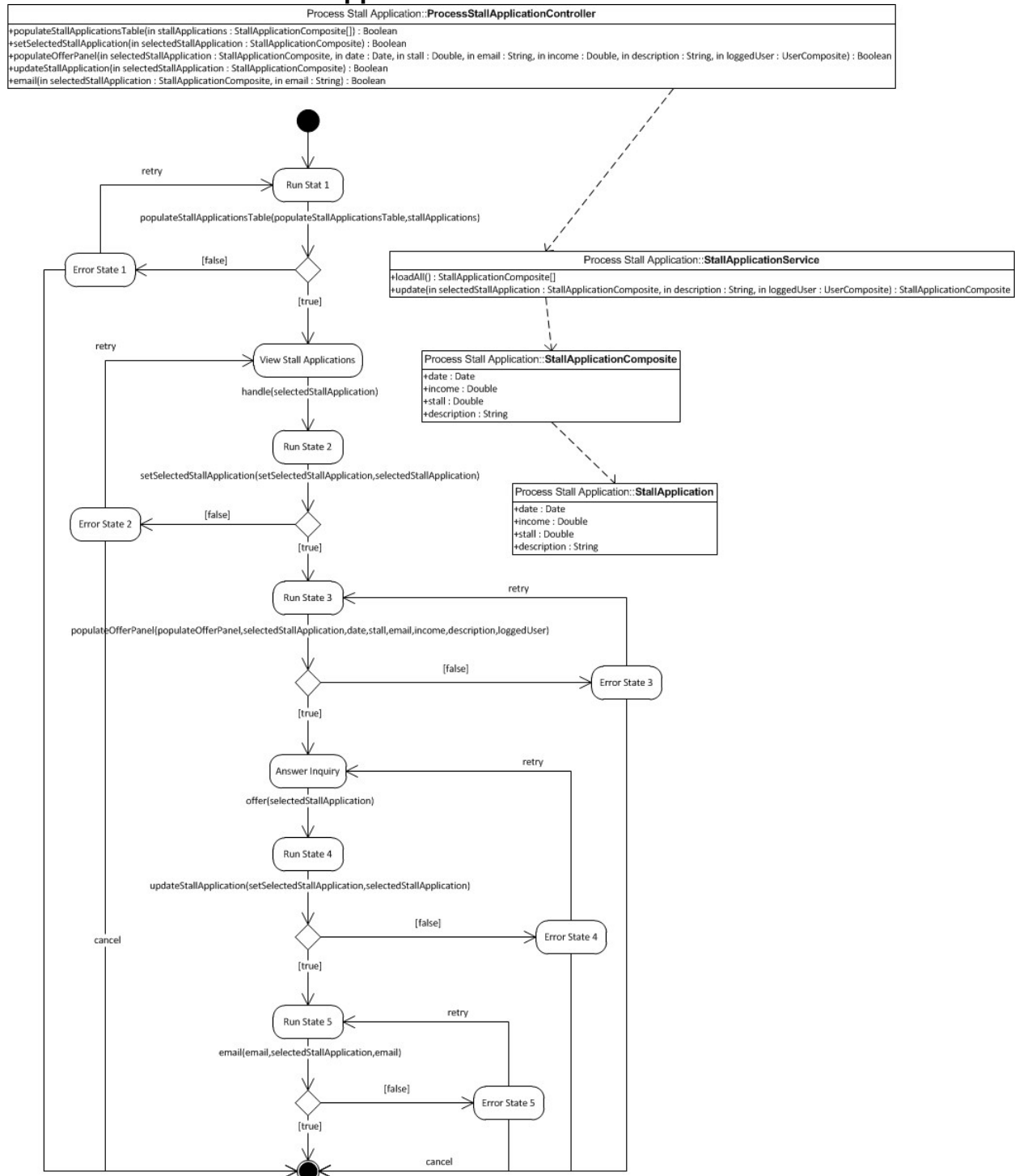


Figure 187 The *apsm* of Process Instalment Application use case

C.2.25 Read News

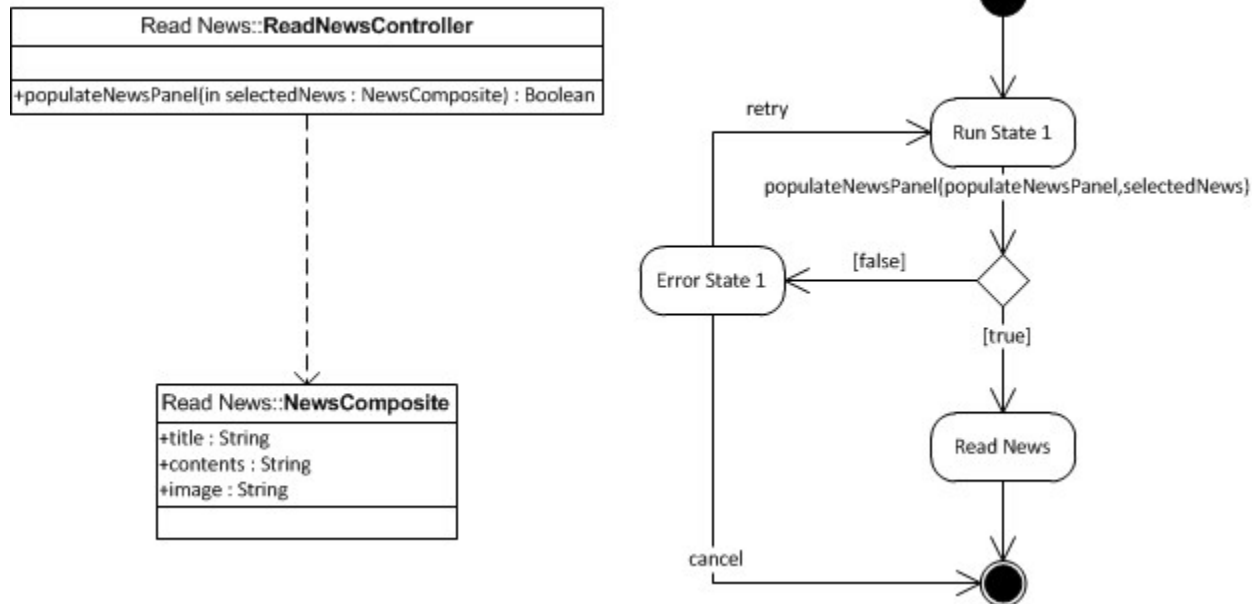


Figure 188 The *apsm* of *Read News* use case

C.2.26 Remove Admin

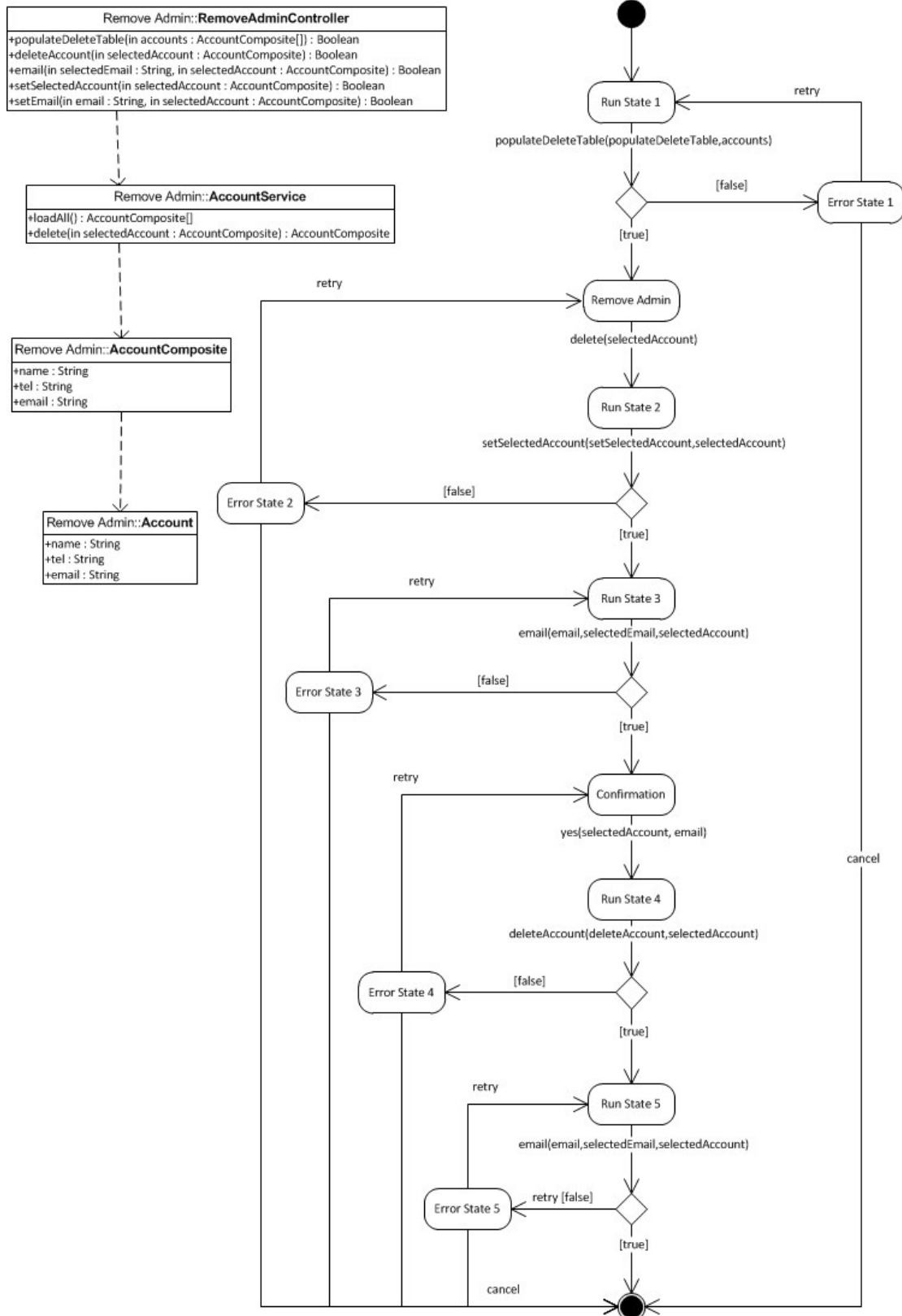


Figure 189 The apsm of Remove Admin use case

C.2.27 Remove Rule

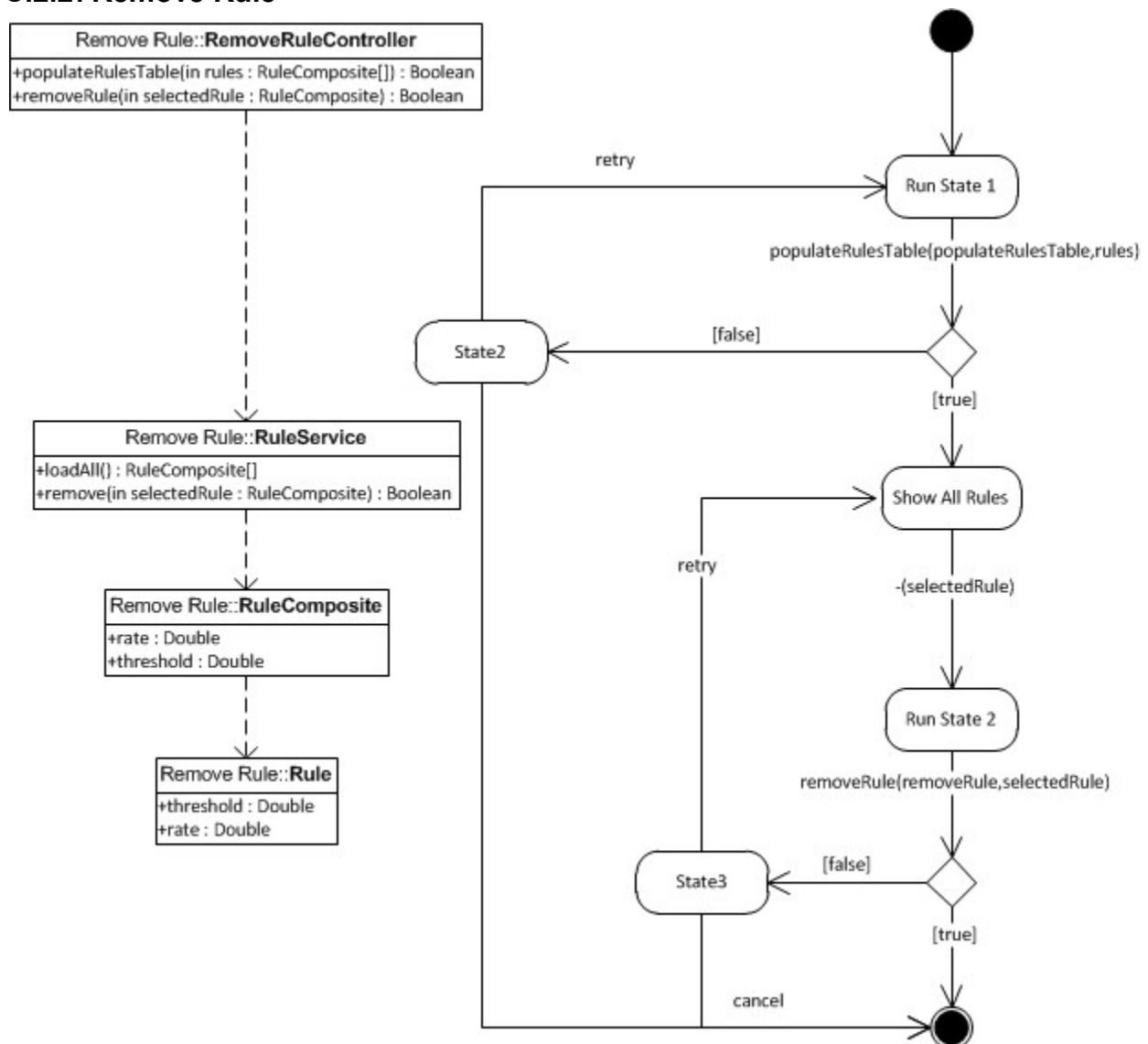


Figure 190 The *apsm* of *Remove Rule* use case

C.2.28 Request Paper/EBill

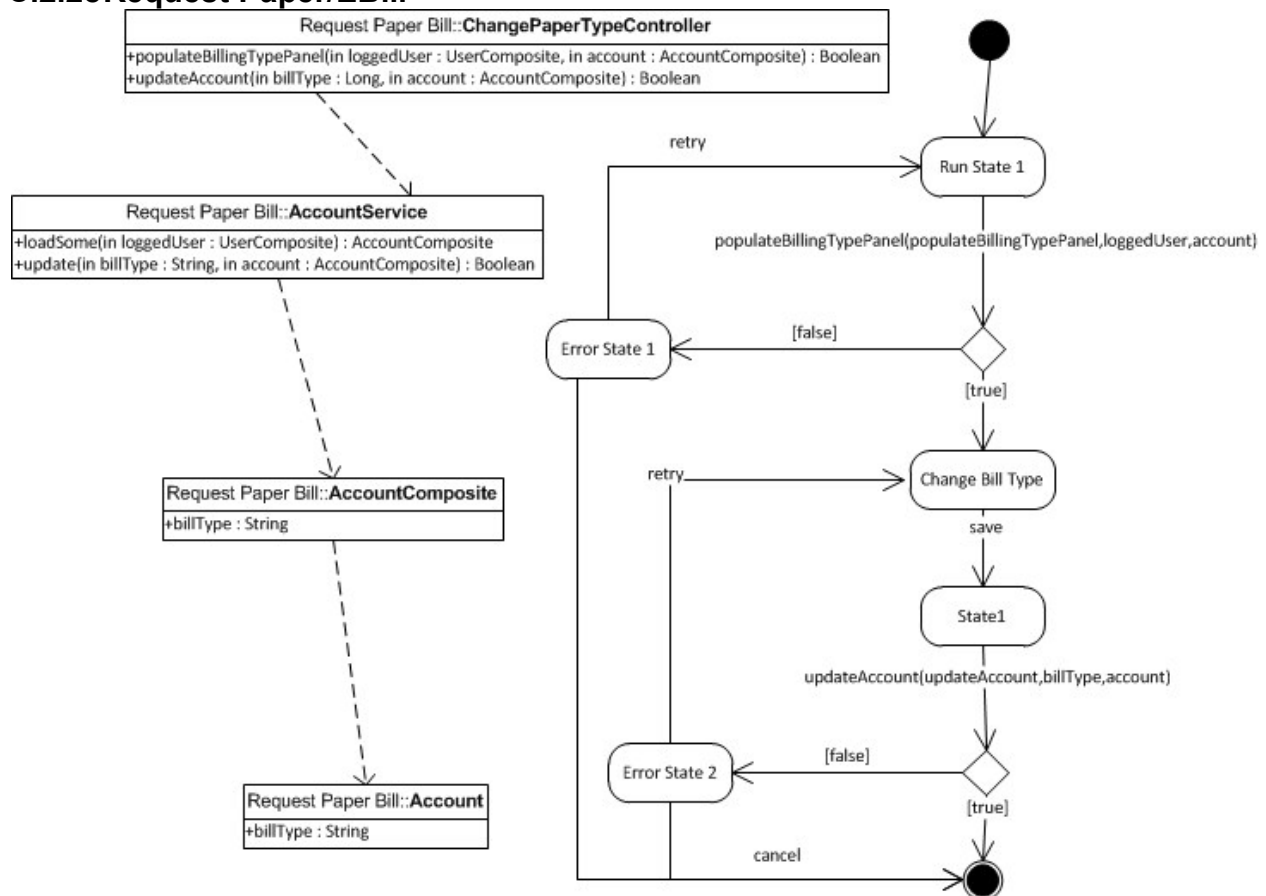


Figure 191 The *apsm* of Request Paper/EBill use case

C.2.29 Apply for Instalments

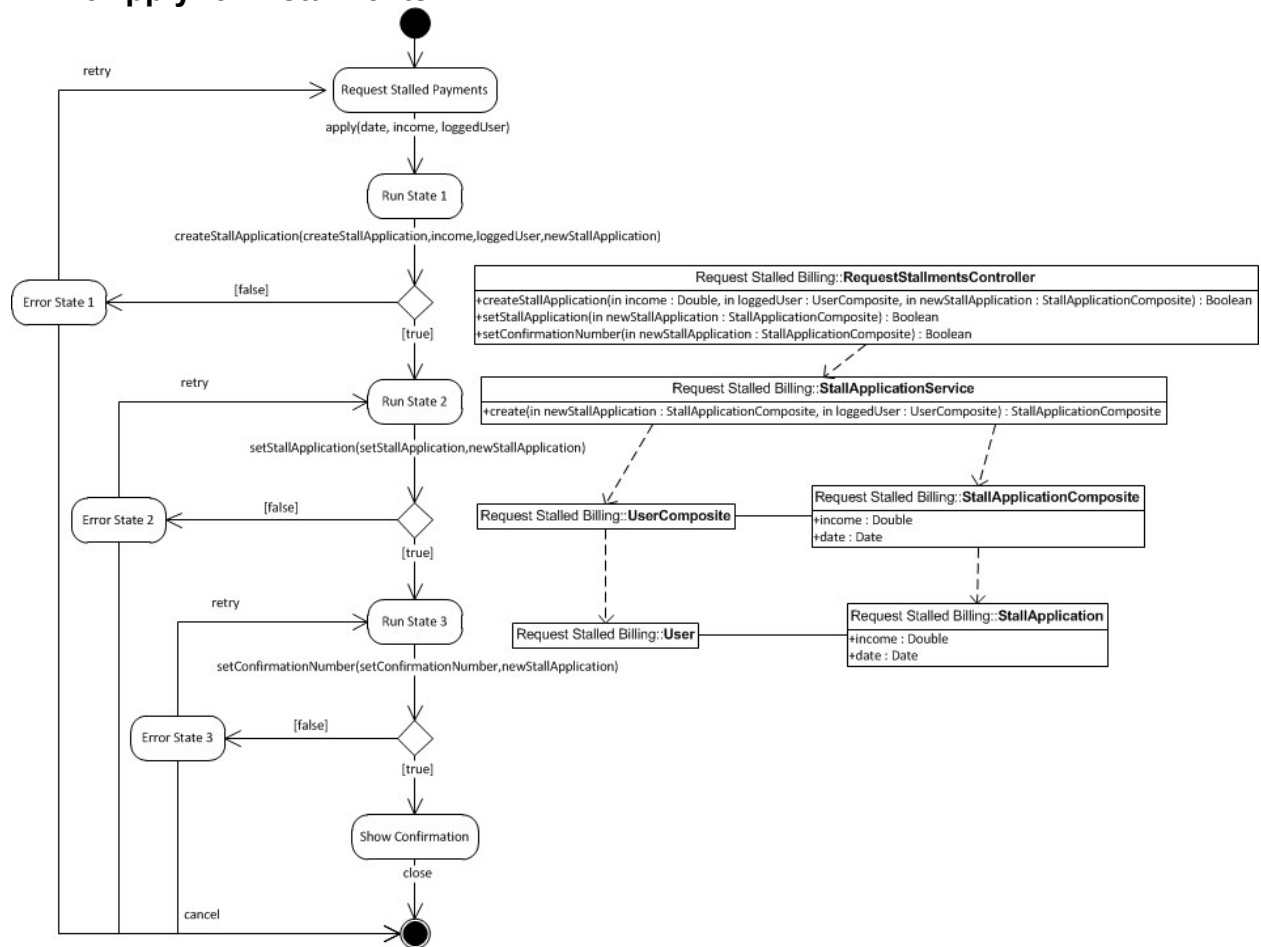


Figure 192 The *apsm* of *Apply for Instalments* use case

C.2.30 Request an Account

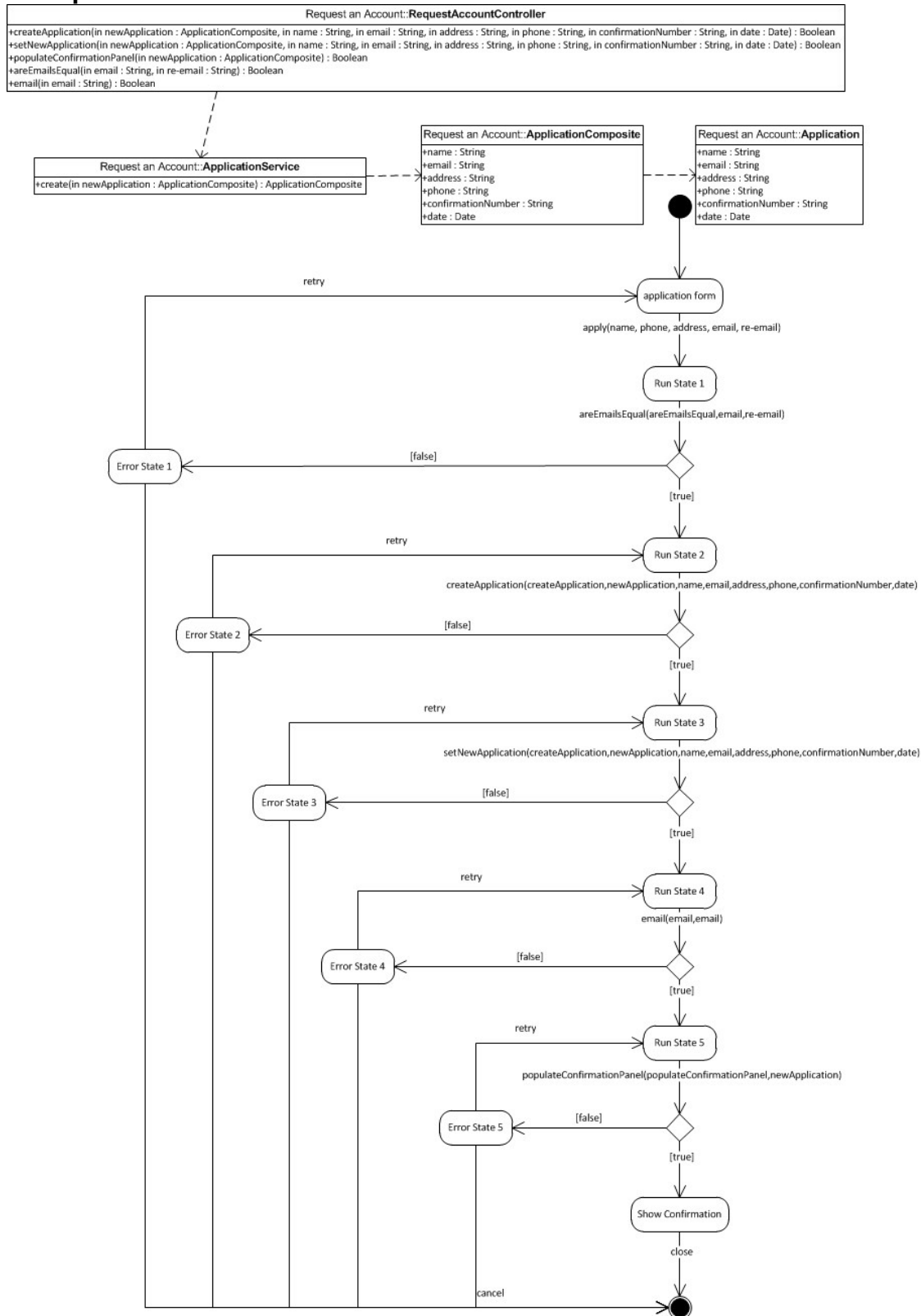


Figure 193 The apsm of Request an Account use case

C.2.31 Request an Appointment

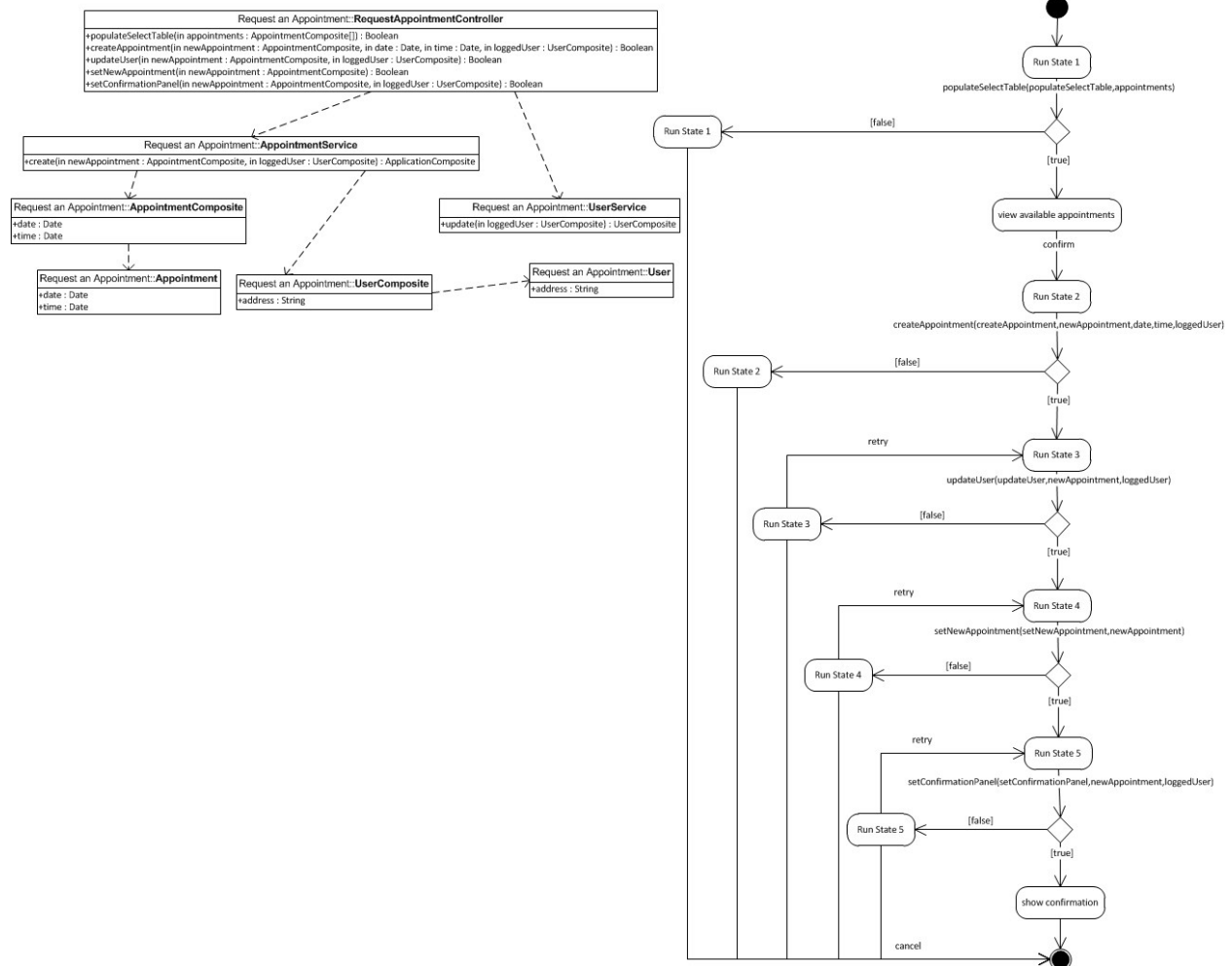


Figure 194 The *apsm* of Request an Appointment use case

C.2.32 Search Tutorial

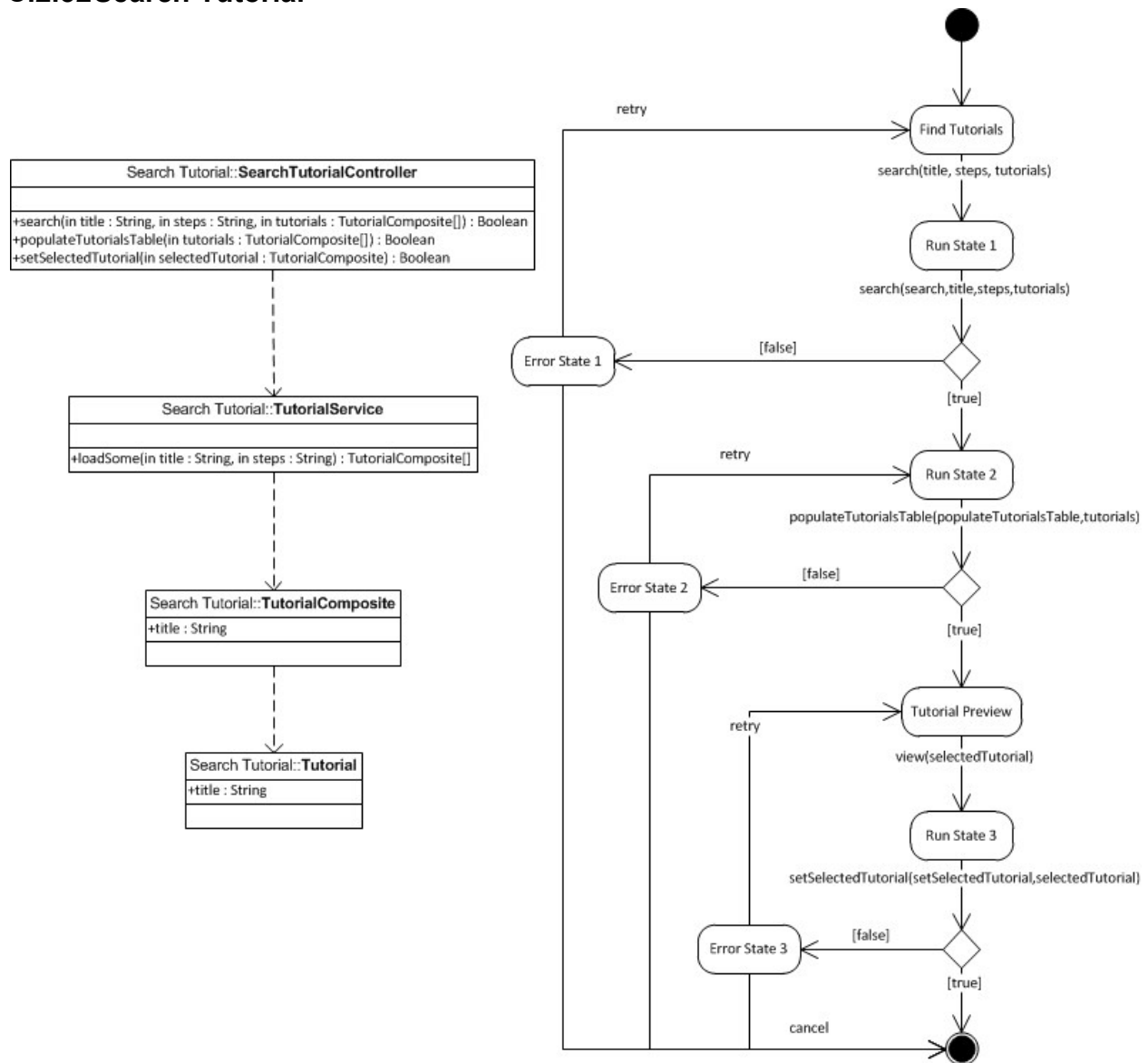


Figure 195 The *apsm* of Search Tutorial use case

C.2.33 View Bill

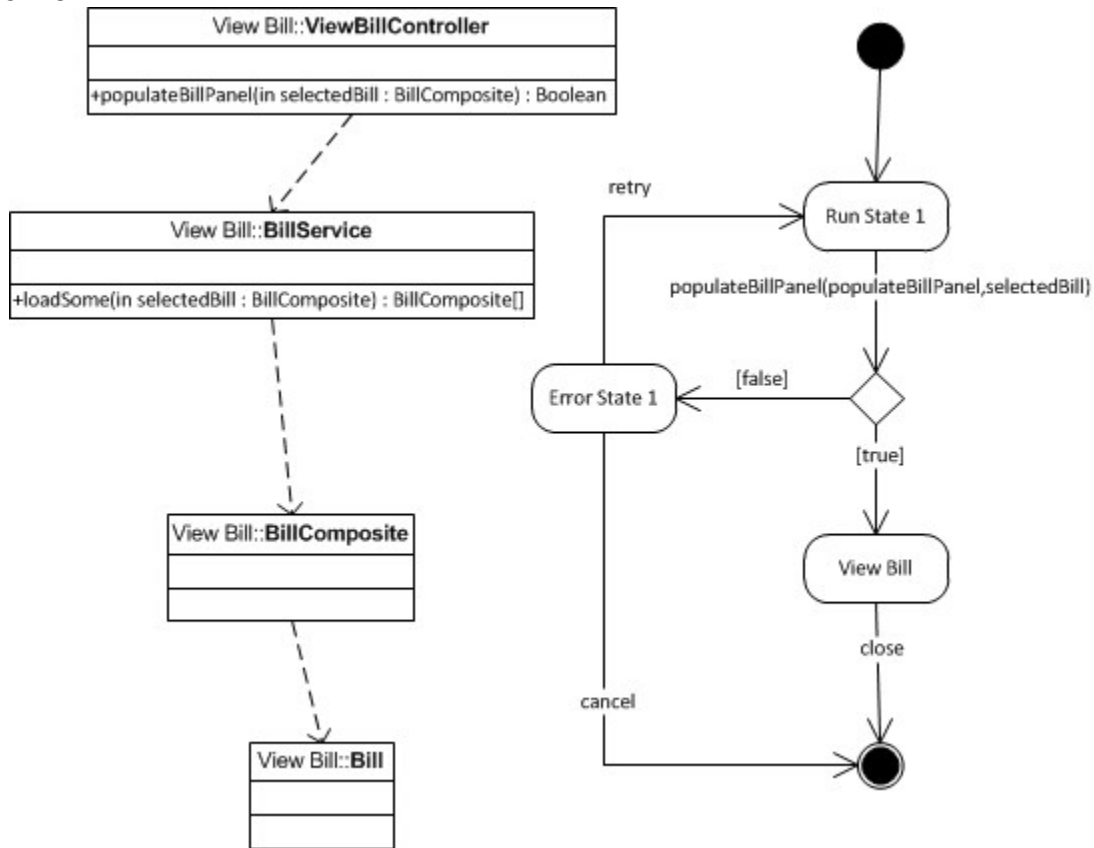


Figure 196 The *apsm* of *View Bill* use case

C.2.34 View Billing Formula

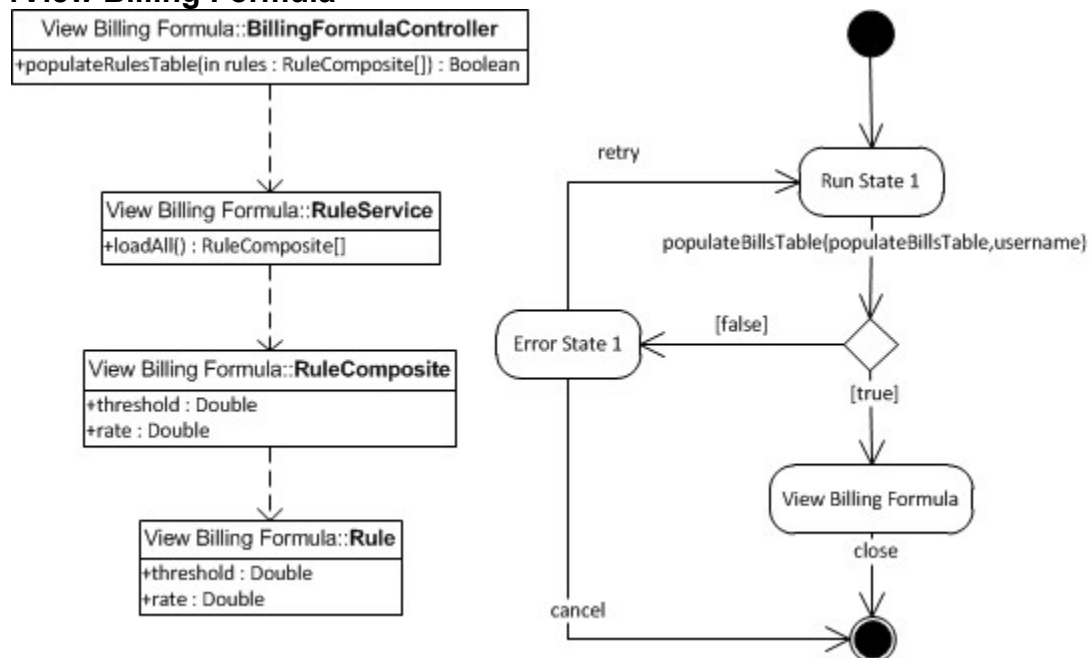


Figure 197 The *apsm* of *View Billing Formula* use case

C.2.35 View Bills

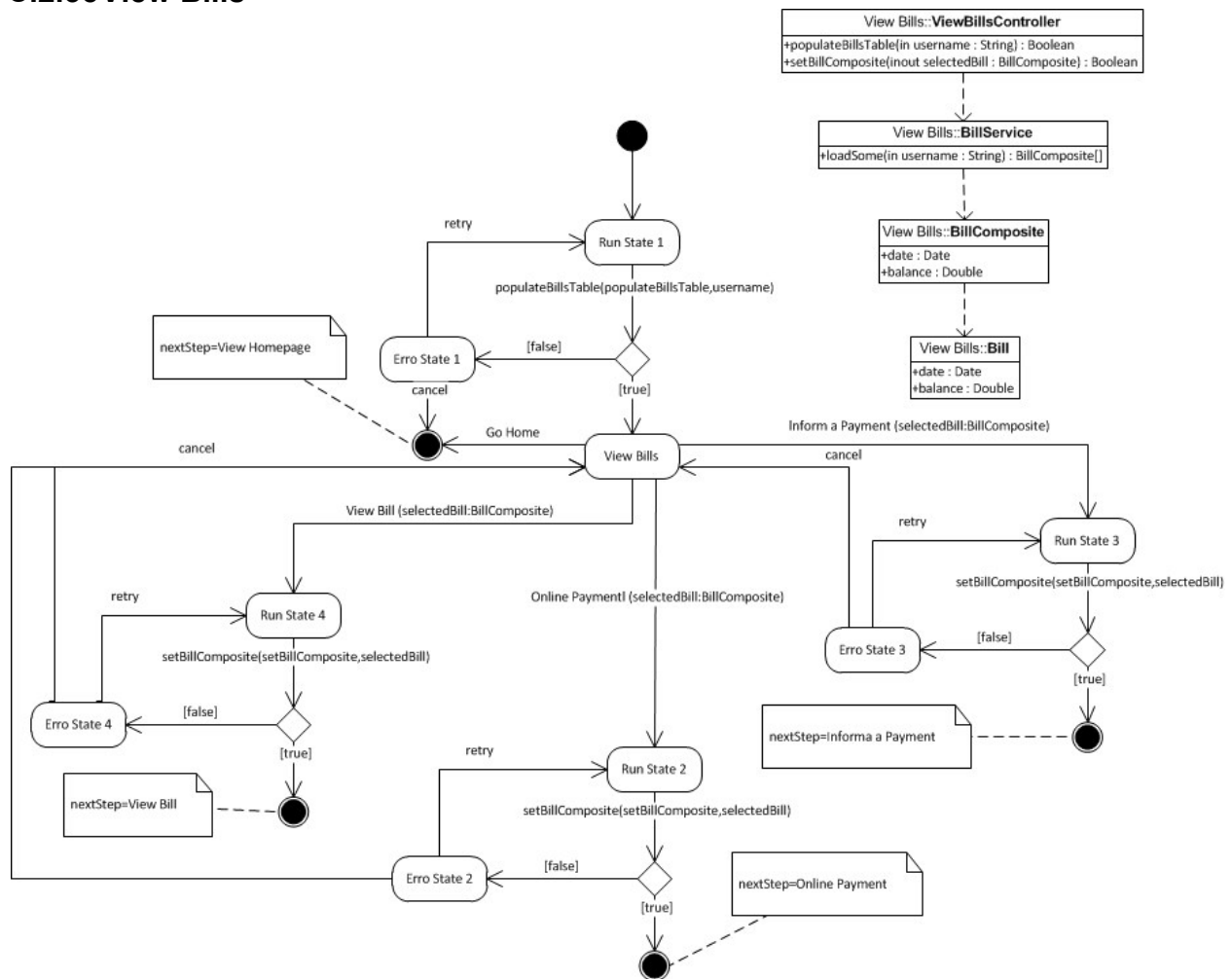


Figure 198 The apsm of View Bills use case

C.2.36 View Tutorial

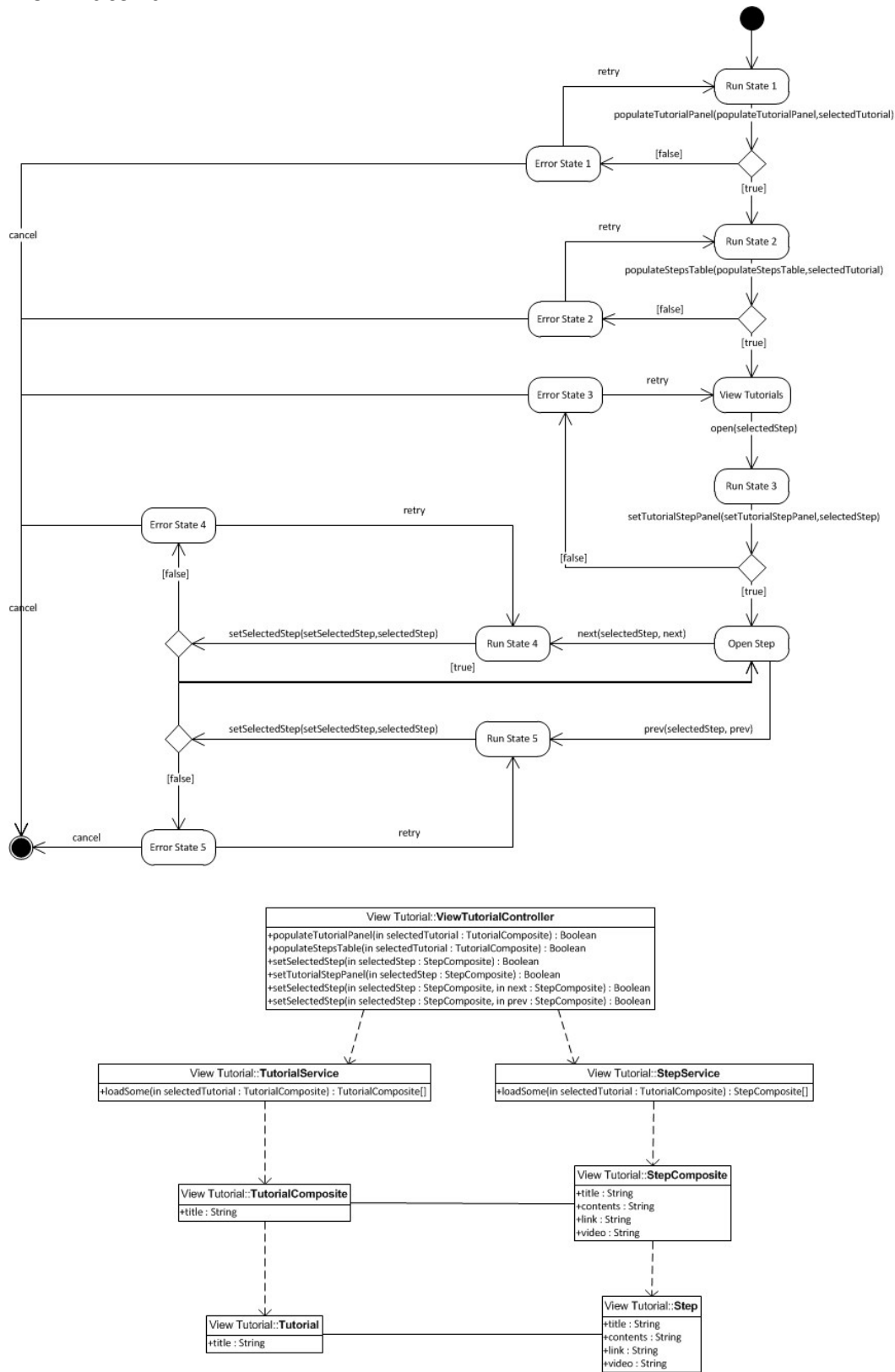


Figure 199 The *apsm* of View Tutorial use case

Appendix D. EMS CASE STUDY

Models of EMS were developed for this thesis - partly presented in Chapter 9. In this appendix, complete listings of the output files created in different steps of MODEWIS applied to EMS –as found in Chapter 9 - are presented.

D.1 *pim*

Listing 19 presents the EMF created by the MODEWIS PIM Generator for the use cases of the EMS in Chapter 9.

Listing 19 The EMF of the pim of three use cases of the EMS

```
<?xml version="1.0" encoding="UTF-8"?>
<wpim:WISMeMoDiagram xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:wpim="http://wpim.modewis"
xsi:schemaLocation="http://wpim.modewis wpim.ecore">
  <hasUseCases name="Login.ems.graphml" application="//@hasApplications.0"
stateMachine="//@hasStateMachines.0" actors="//@hasActors.1"/>
  <hasUseCases name="welcome" application="//@hasApplications.0" stateMachine="//@hasStateMachines.1"
actors="//@hasActors.1"/>
  <hasUseCases name="voting" application="//@hasApplications.0" stateMachine="//@hasStateMachines.2"
actors="//@hasActors.1"/>
  <hasUseCases name="OpenPoll.ems.graphml" application="//@hasApplications.0"
stateMachine="//@hasStateMachines.3" actors="//@hasActors.1"/>
  <hasUseCases name="Voting.ems.graphml" application="//@hasApplications.0"
stateMachine="//@hasStateMachines.4" actors="//@hasActors.0"/>
  <hasUseCases name="homepage" application="//@hasApplications.0" stateMachine="//@hasStateMachines.5"
actors="//@hasActors.1"/>
  <hasApplications name="test" useCases="//@hasUseCases.0 //@hasUseCases.1 //@hasUseCases.2
//@hasUseCases.3 //@hasUseCases.4 //@hasUseCases.5"/>
  <hasStateMachines name="Login.ems.graphml" useCase="//@hasUseCases.0" states="//@hasInitStates.0
//@hasFinalStates.0 //@hasStates.0"/>
  <hasStateMachines name="welcome" useCase="//@hasUseCases.1"/>
  <hasStateMachines name="voting" useCase="//@hasUseCases.2"/>
  <hasStateMachines name="OpenPoll.ems.graphml" useCase="//@hasUseCases.3" states="//@hasInitStates.1
//@hasFinalStates.1 //@hasStates.1 //@hasFinalStates.2"/>
  <hasStateMachines name="Voting.ems.graphml" useCase="//@hasUseCases.4" states="//@hasInitStates.2
//@hasStates.2 //@hasStates.3 //@hasStates.4"/>
  <hasStateMachines name="homepage" useCase="//@hasUseCases.5"/>
  <hasStates name="Login" stateMachine="//@hasStateMachines.0" presentation="//@hasPresentations.1"
incoming="//@hasTransitions.1" outgoing="//@hasTransitions.1"/>
```

```

    <hasStates name="Open Poll" stateMachine="//@hasStateMachines.3" presentation="//@hasPresentations.3"
incoming="//@hasTransitions.4" outgoing="//@hasTransitions.4"/>
    <hasStates name="Enter Voter" stateMachine="//@hasStateMachines.4" presentation="//@hasPresentations.5"
incoming="//@hasTransitions.7" outgoing="//@hasTransitions.7"/>
    <hasStates name="Register Voter (extension point)" stateMachine="//@hasStateMachines.4"/>
    <hasStates name="Show Ballot" stateMachine="//@hasStateMachines.4" presentation="//@hasPresentations.7"
incoming="//@hasTransitions.8" outgoing="//@hasTransitions.8"/>
    <hasPresentations name="welcome" uiComponents="//@hasPresentations.1" holder="//@hasApplications.0"/>
    <hasPresentations name="Login" uiComponents="//@hasOperationTriggers.0"
presentationState="//@hasStates.0" holder="//@hasPresentations.0"/>
    <hasPresentations name="homepage" uiComponents="//@hasPresentations.3" holder="//@hasApplications.0"/>
    <hasPresentations name="Open Poll" uiComponents="//@hasOperationTriggers.1 //@hasOperationTriggers.2"
presentationState="//@hasStates.1" holder="//@hasPresentations.2"/>
    <hasPresentations name="Vote" uiComponents="//@hasPresentations.5" holder="//@hasApplications.0"/>
    <hasPresentations name="Enter Voter" uiComponents="//@hasOperationTriggers.3"
presentationState="//@hasStates.2" holder="//@hasPresentations.4"/>
    <hasPresentations name="Vote" uiComponents="//@hasPresentations.7" holder="//@hasApplications.0"/>
    <hasPresentations name="Show Ballot" uiComponents="//@hasOperationTriggers.4"
presentationState="//@hasStates.4" holder="//@hasPresentations.6"/>
    <hasOperationTriggers name="Login" dataAssociations="//@hasDataAssociations.0 //@hasDataAssociations.1"
uiComponents="//@hasInputFields.0 //@hasPasswordFields.0 //@hasPageVariables.0"
presentation="//@hasPresentations.1" uiAssociations="//@hasDataAssociations.0 //@hasDataAssociations.1"/>
    <hasOperationTriggers name="open" dataAssociations="//@hasDataAssociations.2 //@hasDataAssociations.3
//@hasDataAssociations.4 //@hasDataAssociations.5" uiComponents="//@hasPageVariables.1 //@hasSelectOnes.0
//@hasSelectOnes.1" presentation="//@hasPresentations.3" uiAssociations="//@hasDataAssociations.2
//@hasDataAssociations.3 //@hasDataAssociations.4 //@hasDataAssociations.5"/>
    <hasOperationTriggers name="logout" presentation="//@hasPresentations.3"/>
    <hasOperationTriggers name="Login" dataAssociations="//@hasDataAssociations.6 //@hasDataAssociations.7"
uiComponents="//@hasInputFields.1 //@hasInputFields.2 //@hasDates.0 //@hasPasswordFields.1
//@hasPageVariables.2" presentation="//@hasPresentations.5" uiAssociations="//@hasDataAssociations.6
//@hasDataAssociations.7"/>
    <hasOperationTriggers name="Print" uiComponents="//@hasPageVariables.3"
presentation="//@hasPresentations.7"/>
    <hasDataComposites name="User"/>
    <hasDataComposites name="Poll"/>
    <hasDataComposites name="Election"/>
    <hasDataComposites name="Term"/>
    <hasDataOperations queryType="find"/>
    <hasDataOperations queryType="create"/>
    <hasDataOperations queryType="create"/>
    <hasDataOperations queryType="update"/>
    <hasDataOperations queryType="loadall"/>
    <hasDataOperations queryType="loadsome" parameter="//@hasSelectOnes.0"/>
    <hasDataOperations queryType="find"/>
    <hasInputFields name="Username"/>
    <hasInputFields name="lastname"/>

```



```

<hasInputFields name="firstname"/>
<hasTransitions name="" source="//@hasInitStates.0" target="//@hasStates.0"/>
<hasTransitions name="" source="//@hasStates.0" target="//@hasFinalStates.0"/>
<hasTransitions name="" source="//@hasInitStates.1" target="//@hasStates.1"/>
<hasTransitions name="logout" source="//@hasStates.1" target="//@hasFinalStates.1"/>
<hasTransitions name="open" source="//@hasStates.1" target="//@hasFinalStates.2"/>
<hasTransitions name="not registered" source="//@hasStates.2" target="//@hasStates.3"/>
<hasTransitions name="" source="//@hasInitStates.2" target="//@hasStates.2"/>
<hasTransitions name="login" source="//@hasStates.2" target="//@hasStates.4"/>
<hasTransitions name="print" source="//@hasStates.4"/>
<hasFinalStates name="homepage" stateMachine="//@hasStateMachines.0" incoming="//@hasTransitions.1"
nextStep="//@hasStateMachines.5"/>
<hasFinalStates name="welcome" stateMachine="//@hasStateMachines.3" incoming="//@hasTransitions.3"
nextStep="//@hasStateMachines.1"/>
<hasFinalStates name="voting" stateMachine="//@hasStateMachines.3" incoming="//@hasTransitions.4"
nextStep="//@hasStateMachines.2"/>
<hasInitStates name="startState" stateMachine="//@hasStateMachines.0" outgoing="//@hasTransitions.0"/>
<hasInitStates name="startState" stateMachine="//@hasStateMachines.3" outgoing="//@hasTransitions.2"/>
<hasInitStates name="startState" stateMachine="//@hasStateMachines.4" outgoing="//@hasTransitions.6"/>
<hasDataAssociations next="//@hasDataAssociations.1" isFirst="true" owner="//@hasOperationTriggers.0"
order="0" source="//@hasOperationTriggers.0" target="//@hasDataComposites.0"
dataOperation="//@hasDataOperations.0"/>
<hasDataAssociations owner="//@hasOperationTriggers.0" order="0" source="//@hasDataOperations.0"
target="//@hasPageVariables.0" dataOperation="//@hasDataOperations.1"/>
<hasDataAssociations next="//@hasDataAssociations.3" isFirst="true" owner="//@hasOperationTriggers.1"
order="0" source="//@hasDataOperations.0" target="//@hasPageVariables.0"
dataOperation="//@hasDataOperations.2"/>
<hasDataAssociations next="//@hasDataAssociations.4" owner="//@hasOperationTriggers.1" order="0"
source="//@hasOperationTriggers.1" target="//@hasDataComposites.1"
dataOperation="//@hasDataOperations.3"/>
<hasDataAssociations next="//@hasDataAssociations.5" owner="//@hasOperationTriggers.1" order="0"
source="//@hasDataComposites.2" target="//@hasSelectOnes.0" dataOperation="//@hasDataOperations.4"/>
<hasDataAssociations owner="//@hasOperationTriggers.1" order="0" source="//@hasDataComposites.1"
target="//@hasSelectOnes.0" dataOperation="//@hasDataOperations.5"/>
<hasDataAssociations next="//@hasDataAssociations.7" isFirst="true" owner="//@hasOperationTriggers.3"
order="1" source="//@hasOperationTriggers.3" target="//@hasDataComposites.3"
dataOperation="//@hasDataOperations.6"/>
<hasDataAssociations owner="//@hasOperationTriggers.3" order="2" source="//@hasOperationTriggers.3"
target="//@hasPageVariables.0" dataOperation="//@hasDataOperations.6"/>
<hasActors name="Officer"/>
<hasActors name="Admin"/>
<hasDates name="birthdate"/>
<hasPageVariables name="loggedUser"/>
<hasPageVariables name="theOpenPoll"/>
<hasPageVariables name="voterID"/>
<hasPageVariables name="voterID"/>

```

```

<hasPasswordField name="Password"/>
<hasPasswordField name="pin"/>
<hasSelectOnes name="Elections"/>
<hasSelectOnes name="Polls"/>
</wim:WISMeMoDiagram>

```

D.2 apsm

Listing 20 presents the EMF created by the pim-to-apsm transformations for the use cases of the EMS in Chapter 9.

Listing 20 The EMF of the asm of three use cases of the EMS

```

<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:apsm="http://apsm.modewis" xsi:schemaLocation="http://apsm.modewis apsm.ecore">
  <apsm:Application name="test" useCases="/138 /173 /175 /109 /46 /180"/>
    <apsm:Presentation name="Vote" holder="/0"/>
      <apsm:Presentation name="Show Ballot" holder="/1" presentationState="/21" uiComponents="/3"/>
        <apsm:OperationTrigger name="Print" presentation="/2"/>
          <apsm:Presentation name="Enter Voter" holder="/1" presentationState="/22" uiComponents="/5"/>
            <apsm:OperationTrigger name="Login" uiComponents="/6 /7 /8 /9" presentation="/4"/>
              <apsm:InputField name="lastname"/>
              <apsm:InputField name="firstname"/>
              <apsm>Date name="birthdate"/>
              <apsm:PasswordField name="pin"/>
              <apsm:Presentation name="homepage" holder="/0"/>
              <apsm:Presentation name="Open Poll" holder="/10" presentationState="/24" uiComponents="/12 /25"/>
              <apsm:OperationTrigger name="open" uiComponents="/13 /14" presentation="/11"/>
              <apsm>SelectOne name="Elections"/>
              <apsm>SelectOne name="Polls"/>
              <apsm:Presentation name="welcome" holder="/0"/>
              <apsm:Presentation name="Login" holder="/15" presentationState="/27" uiComponents="/17"/>
              <apsm:OperationTrigger name="Login" uiComponents="/18 /19" presentation="/16"/>
              <apsm:InputField name="Username"/>
              <apsm:PasswordField name="Password"/>
              <apsm:StateMachine name="Voting.ems.graphml" useCase="/46" states="/21 /32 /34 /66 /67 /74 /80 /22
/87"/>
              <apsm:State name="Show Ballot" stateMachine="/20" outgoing="/29"/>
              <apsm:State name="Enter Voter" stateMachine="/20" outgoing="/62" incoming="/86"/>
              <apsm:StateMachine name="OpenPoll.ems.graphml" useCase="/109" states="/24 /98 /100 /113 /119 /123
/178"/>
              <apsm:State name="Open Poll" stateMachine="/23" presentation="/11" outgoing="/31" incoming="/102"/>
              <apsm:OperationTrigger name="logout" presentation="/11"/>
              <apsm:StateMachine name="Login.ems.graphml" useCase="/138" states="/126 /128 /158 /163 /27 /153 /167"/>

```

```

<apism:State name="Login" stateMachine="/26" presentation="/16" outgoing="/147" incoming="/168"/>
<apism:FinalState name="Print" stateMachine="/20" incoming="/29"/>
<apism:Transition name="Print" source="/21" target="/28" stateMachine="/20"/>
<apism:FinalState name="logout" stateMachine="/23" incoming="/31"/>
<apism:Transition name="logout" source="/24" target="/30" stateMachine="/23"/>
<apism:State name="runState1" stateMachine="/20" outgoing="/35"/>
<apism:CallEvent name="call findTerm()" operation="/44" owner="/35"/>
<apism:Choice name="choice1" stateMachine="/20" outgoing="/36 /88" incoming="/35"/>
<apism:Transition source="/32" target="/34" callEvents="/33" stateMachine="/20"/>
<apism:Transition source="/34" target="/66" stateMachine="/20" guard="/72"/>
<apism:DataEntity name="Term" attributes="/38 /51 /54 /58" dataComposite="/41"/>
<apism:Attribute name="lastname" owningClass="/37" type="/39"/>
<apism:Type name="string"/>
<apism:Attribute name="lastname" owningClass="/41" type="/39"/>
<apism:DataComposite name="Term" attributes="/40 /52 /56 /59" service="/48" dataEntities="/37"/>
<apism:Type primitiveType="boolean" name="boolean"/>
<apism:Controller name="Voting.ems.graphmlController" operations="/44" useCase="/46"/>
<apism:Operation name="findTerm" parameters="/45 /53 /57 /60 /61" callee="/49" class="/43"
returnType="/42"/>
<apism:Parameter name="lastname" type="/39" behavior="/44"/>
<apism:UseCase name="Voting.ems.graphml" application="/0" stateMachine="/20" controller="/43"
actors="/179"/>
<apism:Type primitiveType="unknown" auxiliaryType="/41" name="Term"/>
<apism:Service name="TermService" operations="/49" dataComposite="/41"/>
<apism:Operation name="findTerm" parameters="/50" crudNature="find" class="/48"/>
<apism:Parameter name="Term" type="/47" behavior="/49"/>
<apism:Attribute name="firstname" owningClass="/37" type="/39"/>
<apism:Attribute name="firstname" owningClass="/41" type="/39"/>
<apism:Parameter name="firstname" type="/39" behavior="/44"/>
<apism:Attribute name="birthdate" owningClass="/37" type="/55"/>
<apism:Type primitiveType="date" name="date"/>
<apism:Attribute name="birthdate" owningClass="/41" type="/55"/>
<apism:Parameter name="birthdate" type="/39" behavior="/44"/>
<apism:Attribute name="pin" owningClass="/37" type="/39"/>
<apism:Attribute name="pin" owningClass="/41" type="/39"/>
<apism:Parameter name="pin" type="/39" behavior="/44"/>
<apism:Parameter name="voterID" type="/47" behavior="/44"/>
<apism:Transition source="/22" target="/32" signalEvent="/63" stateMachine="/20"/>
<apism:SignalEvent name="Login" parameters="/64 /65 /95 /96 /97" transition="/62"/>
<apism:Parameter name="lastname" type="/39" behavior="/63"/>
<apism:Parameter name="firstname" type="/39" behavior="/63"/>
<apism:State name="runState2" stateMachine="/20" outgoing="/68" incoming="/36 /69"/>
<apism:Choice name="choice2" stateMachine="/20" outgoing="/69 /71 /75" incoming="/68"/>
<apism:Transition source="/66" target="/67" callEvents="/70" stateMachine="/20"/>
<apism:Transition source="/67" target="/66" stateMachine="/20" guard="/72"/>
<apism:CallEvent owner="/68"/>

```

```

<apism:Transition source="/67" stateMachine="/20" guard="/72"/>
<apism:Guard name="true"/>
<apism:Guard name="false"/>
<apism:State name="Error State 2" stateMachine="/20" presentation="/76" outgoing="78 /81"
incoming="75"/>
<apism:Transition source="/67" target="/74" stateMachine="/20" guard="/73"/>
<apism:Presentation name="Error Page 2" uiComponents="/77 /84"/>
<apism:StaticText name="Error Performing Operation 2"/>
<apism:Transition source="/74" target="/66" signalEvent="/79" stateMachine="/20"/>
<apism:SignalEvent name="retry" transition="/78"/>
<apism:FinalState name="welcome" stateMachine="/20" incoming="81 /93" nextStep="/83"/>
<apism:Transition source="/74" target="/80" signalEvent="/82" stateMachine="/20"/>
<apism:SignalEvent name="cancel" transition="/81"/>
<apism:StateMachine name="welcome" useCase="/173"/>
<apism:StaticText name="Error Performing Operation 2"/>
<apism:StartState name="startState" stateMachine="/20" outgoing="86"/>
<apism:Transition source="/85" target="/22" stateMachine="/20"/>
<apism:State name="Error State 1" stateMachine="/20" presentation="/89" outgoing="91 /93"
incoming="88"/>
<apism:Transition source="/34" target="/87" stateMachine="/20" guard="/73"/>
<apism:Presentation name="Error Page 1" uiComponents="/90"/>
<apism:StaticText name="Error Performing Operation 1"/>
<apism:Transition source="/87" target="/32" signalEvent="/92" stateMachine="/20"/>
<apism:SignalEvent name="retry" transition="/91"/>
<apism:Transition source="/87" target="/80" signalEvent="/94" stateMachine="/20"/>
<apism:SignalEvent name="cancel" transition="/93"/>
<apism:Parameter name="birthdate" type="/39" behavior="/63"/>
<apism:Parameter name="pin" type="/39" behavior="/63"/>
<apism:Parameter name="voterID" type="/47" behavior="/63"/>
<apism:State name="runState0" stateMachine="/23" outgoing="101" incoming="102 /110 /124"/>
<apism:CallEvent name="call setLoggedInUser()" operation="/107" owner="/101"/>
<apism:Choice name="choice0" stateMachine="/23" outgoing="102 /110 /112 /114" incoming="101"/>
<apism:Transition source="/98" target="/100" callEvents="/99 /111" stateMachine="/23"/>
<apism:Transition source="/100" target="/98" stateMachine="/23" guard="/72"/>
<apism:Type primitiveType="unknown" auxiliaryType="/104" name="Poll"/>
<apism:DataComposite name="Poll"/>
<apism:Parameter name="foundPoll" type="/103"/>
<apism:Controller name="OpenPoll.ems.graphmlController" operations="/107" useCase="/109"/>
<apism:Operation name="setLoggedInUser" parameters="/105 /108" class="/106" returnType="/42"/>
<apism:Parameter name="loggedInUser" type="/103" behavior="/107"/>
<apism:UseCase name="OpenPoll.ems.graphml" application="/0" stateMachine="/23" controller="/106"
actors="/172"/>
<apism:Transition source="/100" target="/98" stateMachine="/23" guard="/72"/>
<apism:CallEvent owner="/101"/>
<apism:Transition source="/100" stateMachine="/23" guard="/72"/>

```

```

    <apism:State name="Error State 0" stateMachine="/23" presentation="/115" outgoings="/117 /120"
incomings="/114"/>
    <apism:Transition source="/100" target="/113" stateMachine="/23" guard="/73"/>
    <apism:Presentation name="Error Page 0" uiComponents="/116 /122 /125 /160 /166 /169"/>
    <apism:StaticText name="Error Performing Operation 0"/>
    <apism:Transition source="/113" target="/98" signalEvent="/118" stateMachine="/23"/>
    <apism:SignalEvent name="retry" transition="/117"/>
    <apism:FinalState name="welcome" stateMachine="/23" incomings="/120" nextStep="/83"/>
    <apism:Transition source="/113" target="/119" signalEvent="/121" stateMachine="/23"/>
    <apism:SignalEvent name="cancel" transition="/120"/>
    <apism:StaticText name="Error Performing Operation 0"/>
    <apism:StartState name="startState" stateMachine="/23" outgoings="/124"/>
    <apism:Transition source="/123" target="/98"/>
    <apism:StaticText name="Error Performing Operation 0"/>
    <apism:State name="runState0" stateMachine="/26" outgoings="/129" incomings="/130"/>
    <apism:CallEvent name="call findUser()" operation="/136" owner="/129"/>
    <apism:Choice name="choice0" stateMachine="/26" outgoings="/130 /151 /152 /159" incomings="/129"/>
    <apism:Transition source="/126" target="/128" callEvents="/127 /154" stateMachine="/26"/>
    <apism:Transition source="/128" target="/126" stateMachine="/26" guard="/72"/>
    <apism:DataEntity name="User" attributes="/132 /143" dataComposite="/134"/>
    <apism:Attribute name="username" owningClass="/131" type="/39"/>
    <apism:Attribute name="username" owningClass="/134" type="/39"/>
    <apism:DataComposite name="User" attributes="/133 /144" service="/140" dataEntities="/131"/>
    <apism:Controller name="Login.ems.graphmlController" operations="/136 /156" useCase="/138"/>
    <apism:Operation name="findUser" parameters="/137 /145 /146" callee="/141" class="/135"
returnType="/42"/>
    <apism:Parameter name="Username" type="/39" behavior="/136"/>
    <apism:UseCase name="Login.ems.graphml" application="/0" stateMachine="/26" controller="/135"
actors="/172"/>
    <apism:Type primitiveType="unknown" auxiliaryType="/134" name="User"/>
    <apism:Service name="UserService" operations="/141" dataComposite="/134"/>
    <apism:Operation name="findUser" parameters="/142" crudNature="find" class="/140"/>
    <apism:Parameter name="User" type="/139" behavior="/141"/>
    <apism:Attribute name="password" owningClass="/131" type="/39"/>
    <apism:Attribute name="password" owningClass="/134" type="/39"/>
    <apism:Parameter name="Password" type="/39" behavior="/136"/>
    <apism:Parameter name="loggedUser" type="/139" behavior="/136"/>
    <apism:Transition source="/27" target="/126" signalEvent="/148" stateMachine="/26"/>
    <apism:SignalEvent name="Login" parameters="/149 /150 /170" transition="/147"/>
    <apism:Parameter name="Username" type="/39" behavior="/148"/>
    <apism:Parameter name="Password" type="/39" behavior="/148"/>
    <apism:Transition source="/128" stateMachine="/26" guard="/72"/>
    <apism:Transition source="/128" target="/153" stateMachine="/26" guard="/72"/>
    <apism:FinalState name="homepage" stateMachine="/26" incomings="/152" nextStep="/171"/>
    <apism:CallEvent name="call setLoggedUser()" operation="/156" owner="/129"/>
    <apism:Parameter name="foundUser" type="/139"/>

```

```

<apsm:Operation name="setLoggedUser" parameters="/155 /157" class="/135" returnType="/42"/>
<apsm:Parameter name="loggedUser" type="/139" behavior="/156"/>
<apsm:State name="Error State 0" stateMachine="/26" presentation="/115" outgoings="/161 /164"
incomings="/159"/>
<apsm:Transition source="/128" target="/158" stateMachine="/26" guard="/73"/>
<apsm:StaticText name="Error Performing Operation 0"/>
<apsm:Transition source="/158" target="/126" signalEvent="/162" stateMachine="/26"/>
<apsm:SignalEvent name="retry" transition="/161"/>
<apsm:FinalState name="welcome" stateMachine="/26" incomings="/164" nextStep="/83"/>
<apsm:Transition source="/158" target="/163" signalEvent="/165" stateMachine="/26"/>
<apsm:SignalEvent name="cancel" transition="/164"/>
<apsm:StaticText name="Error Performing Operation 0"/>
<apsm:StartState name="startState" stateMachine="/26" outgoings="/168"/>
<apsm:Transition source="/167" target="/27" stateMachine="/26"/>
<apsm:StaticText name="Error Performing Operation 0"/>
<apsm:Parameter name="loggedUser" type="/139" behavior="/148"/>
<apsm:StateMachine name="homepage" useCase="/180"/>
<apsm:Actor name="Admin" useCases="/138 /173 /175 /109 /180"/>
<apsm:UseCase name="welcome" application="/0" stateMachine="/83" controller="/174" actors="/172"/>
<apsm:Controller name="welcomeController"/>
<apsm:UseCase name="voting" application="/0" stateMachine="/176" controller="/177" actors="/172"/>
<apsm:StateMachine name="voting" useCase="/175"/>
<apsm:Controller name="votingController"/>
<apsm:FinalState name="voting" stateMachine="/23" nextStep="/176"/>
<apsm:Actor name="Officer" useCases="/46"/>
<apsm:UseCase name="homepage" application="/0" stateMachine="/171" controller="/181" actors="/172"/>
<apsm:Controller name="homepageController"/>
</xmi:XMI>

```

D.3 gwt

Listing 21 presents the EMF created by the apsm-to-gwt transformations for the use cases of the EMS in Chapter 9.

Listing 21 The EMF of the gwt of three use cases of the EMS

```

<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:gwt="http://gwt.modewis" xsi:schemaLocation="http://gwt.modewis gwt.ecore">
  <gwt:Application name="test" mainEntryPoint="/104" client="/1" shared="/27" server="/58"/>
  <gwt:Package name="client" classes="/4 /3 /5 /6 /7 /8 /9 /2 /20 /19 /21 /23 /22 /65 /67 /84 /86 /104"
packages="/87 /30"/>
  <gwt:Panel name="rootPanel" owningPackage="/1" container="/2" widgets="/3 /10 /14 /19 /22"/>
  <gwt:Panel name="Error Page 0Panel" owningPackage="/1" container="/2" widgets="/4 /5 /6 /7 /8 /9"/>
  <gwt:Label owningPackage="/1" container="/3"/>

```

```

<gwt:Label owningPackage="/1" container="/3"/>
<gwt:Label owningPackage="/1" container="/3"/>
<gwt:Label owningPackage="/1" container="/3"/>
<gwt:Label owningPackage="/1" container="/3"/>
<gwt:Label owningPackage="/1" container="/3"/>
<gwt:Panel name="LoginPanel" owningPackage="/11" container="/2" widgets="/12"/>
<gwt:Package name="Login.ems.graphml" owningPackage="/1" classes="/4 /3 /5 /6 /7 /8 /9 /2 /10 /12 /24
/68 /75"/>
<gwt:Panel name="LoginForm" owningPackage="/11" container="/10" widgets="/13 /24 /100"/>
<gwt:PasswordTextBox name="Password" owningPackage="/11" container="/12"/>
<gwt:Panel name="Open Poll" owningPackage="/15" container="/2" widgets="/16"/>
<gwt:Package name="OpenPoll.ems.graphml" owningPackage="/1" classes="/4 /3 /5 /6 /7 /8 /9 /2 /20 /19 /21
/23 /22 /14"/>
<gwt:Panel name="OpenPanel" owningPackage="/15" container="/14" widgets="/17 /18"/>
<gwt:ListBox owningPackage="/15" container="/16"/>
<gwt:ListBox owningPackage="/15" container="/16"/>
<gwt:Panel name="Error Page 2Panel" owningPackage="/1" container="/2" widgets="/20 /21"/>
<gwt:Label owningPackage="/1" container="/19"/>
<gwt:Label owningPackage="/1" container="/19"/>
<gwt:Panel name="Error Page 1Panel" owningPackage="/1" container="/2" widgets="/23"/>
<gwt:Label owningPackage="/1" container="/22"/>
<gwt:TextBox name="Username" owningPackage="/11" container="/12"/>
<gwt:Type dataType="string"/>
<gwt:Type dataType="string"/>
<gwt:Package name="shared" classes="/28 /40 /62 /81"/>
<gwt:Class name="UserComposite" owningPackage="/27" clientDependencies="/32" attributes="/29 /36"/>
<gwt:Attribute name="password" owningClass="/28" type="/25"/>
<gwt:Package name="domain" owningPackage="/1" classes="/31 /42"/>
<gwt:Class name="User" owningPackage="/30" supplierDependencies="/32" attributes="/33 /37"/>
<gwt:Dependency client="/28" supplier="/31"/>
<gwt:Attribute name="password" owningClass="/31" type="/26"/>
<gwt:Type dataType="string"/>
<gwt:Type dataType="string"/>
<gwt:Attribute name="username" owningClass="/28" type="/34"/>
<gwt:Attribute name="username" owningClass="/31" type="/35"/>
<gwt:Type dataType="string"/>
<gwt:Type dataType="string"/>
<gwt:Class name="TermComposite" owningPackage="/27" clientDependencies="/43" attributes="/41 /47 /51
/55"/>
<gwt:Attribute name="pin" owningClass="/40" type="/38"/>
<gwt:Class name="Term" owningPackage="/30" supplierDependencies="/43" attributes="/44 /48 /52 /56"/>
<gwt:Dependency client="/40" supplier="/42"/>
<gwt:Attribute name="pin" owningClass="/42" type="/39"/>
<gwt:Type dataType="date"/>
<gwt:Type dataType="date"/>
<gwt:Attribute name="birthdate" owningClass="/40" type="/45"/>

```

```

<gwt:Attribute name="birthdate" owningClass="/42" type="/46"/>
<gwt:Type dataType="string"/>
<gwt:Type dataType="string"/>
<gwt:Attribute name="firstname" owningClass="/40" type="/49"/>
<gwt:Attribute name="firstname" owningClass="/42" type="/50"/>
<gwt:Type dataType="string"/>
<gwt:Type dataType="string"/>
<gwt:Attribute name="lastname" owningClass="/40" type="/53"/>
<gwt:Attribute name="lastname" owningClass="/42" type="/54"/>
<gwt:Type classType="/62"/>
<gwt:Package name="server" classes="/59 /78"/>
<gwt:ServiceImpl name="UserServiceImpl" owningPackage="/58" clientDependencies="/70" implements="/65"
operations="/60"/>
<gwt:Operation name="findUser" class="/59" returnType="/63" parameters="/61"/>
<gwt:Parameter name="User" type="/57"/>
<gwt:Class name="User" owningPackage="/27"/>
<gwt:Type dataType="void" classType="/62"/>
<gwt:Parameter name="User" type="/57"/>
<gwt:Service name="UserService" owningPackage="/1" operations="/66" asyncService="/67"/>
<gwt:Operation name="findUser" class="/65" returnType="/63" parameters="/64"/>
<gwt:ServiceAsync name="UserService" owningPackage="/1" operations="/71"/>
<gwt:Class name="Login.ems.graphmlController" owningPackage="/11" supplierDependencies="/70"
operations="/69 /98 /101"/>
<gwt:Operation name="findUser" class="/68" returnType="/97" callee="/60"/>
<gwt:Dependency client="/68" supplier="/59"/>
<gwt:Operation name="findUser" class="/67" returnType="/76" parameters="/72 /73"/>
<gwt:Parameter name="User" type="/63"/>
<gwt:Parameter name="findUserCallback" type="/74"/>
<gwt:Type classType="/75"/>
<gwt:AsyncCallback name="FindUserCallback" owningPackage="/11" parameter="/62"/>
<gwt:Type dataType="void"/>
<gwt:Type classType="/81"/>
<gwt:ServiceImpl name="TermServiceImpl" owningPackage="/58" clientDependencies="/90" implements="/84"
operations="/79"/>
<gwt:Operation name="findTerm" class="/78" returnType="/82" parameters="/80"/>
<gwt:Parameter name="Term" type="/77"/>
<gwt:Class name="Term" owningPackage="/27"/>
<gwt:Type dataType="void" classType="/81"/>
<gwt:Parameter name="Term" type="/77"/>
<gwt:Service name="TermService" owningPackage="/1" operations="/85" asyncService="/86"/>
<gwt:Operation name="findTerm" class="/84" returnType="/82" parameters="/83"/>
<gwt:ServiceAsync name="TermService" owningPackage="/1" operations="/91"/>
<gwt:Package name="Voting.ems.graphml" owningPackage="/1" classes="/88 /95"/>
<gwt:Class name="Voting.ems.graphmlController" owningPackage="/87" supplierDependencies="/90"
operations="/89"/>
<gwt:Operation name="findTerm" class="/88" returnType="/82" callee="/79"/>

```



```

<gwt:Dependency client="/88" supplier="/78"/>
<gwt:Operation name="findTerm" class="/86" returnType="/96" parameters="/92 /93"/>
<gwt:Parameter name="Term" type="/82"/>
<gwt:Parameter name="findTermCallback" type="/94"/>
<gwt:Type classType="/95"/>
<gwt:AsyncCallback name="FindTermCallback" owningPackage="/87" parameter="/81"/>
<gwt:Type dataType="void"/>
<gwt:Type dataType="void"/>
<gwt:Operation name="onClick" class="/68" returnType="/103" callee="/101"/>
<gwt:Type dataType="void"/>
<gwt:Button name="LoginButton" owningPackage="/11" eventHandlers="/68" container="/10"/>
<gwt:Operation name="setLoggedUser" class="/68" returnType="/102"/>
<gwt:Type dataType="void"/>
<gwt:Type dataType="void"/>
<gwt:MainEntryPoint name="testEntryPoint" owningPackage="/1"/>
</xmi:XMI>

```

D.4 dotnet

Listing 22 presents the EMF created by the apsm-to-dotnet transformations for the use cases of the EMS in Chapter 9.

Listing 22 The EMF of the dotnet of three use cases of the EMS

```

<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:dotnet="http://dotnet.modewis" xsi:schemaLocation="http://dotnet.modewis dotnet.ecore">
  <dotnet:Database name="test" tables="/8" storedProcedures="/9"/>
    <dotnet:Table name="User" owningFolder="/0"/>
      <dotnet:StoredProcedure name="findUser" owningFolder="/0" parameters="/36 /39" queryType="find"/>
      <dotnet:Application name="test" files="/23 /26 /29" folders="/4 /21 /24 /27 /79 /81 /84" appCode="/4"/>
      <dotnet:Folder name="App_Code" owningFolder="/3" files="/5 /6 /7 /10 /11 /12 /31 /41 /78 /80 /82 /83"/>
      <dotnet>DataComposite name="User" owningFolder="/4" attributes="/13 /15" service="/6"/>
      <dotnet:DataService name="UserService" owningFolder="/4" operations="/33" dataComposite="/5"
dataAccess="/7"/>
      <dotnet:DataAccess name="UserAccess" owningFolder="/4" operations="/37"/>
      <dotnet:Table name="Term" owningFolder="/0"/>
      <dotnet:StoredProcedure name="findTerm" owningFolder="/0" parameters="/46 /49 /51 /53"
queryType="find"/>
      <dotnet>DataComposite name="Term" owningFolder="/4" attributes="/16 /17 /19 /20" service="/11"/>
      <dotnet:DataService name="TermService" owningFolder="/4" operations="/43" dataComposite="/10"
dataAccess="/12"/>
      <dotnet:DataAccess name="TermAccess" owningFolder="/4" operations="/47"/>
      <dotnet:Attribute name="password" owningClass="/5" type="/14"/>
      <dotnet:Type primitiveType="string"/>

```

```

<dotnet:Attribute name="username" owningClass="/5" type="/14"/>
<dotnet:Attribute name="pin" owningClass="/10" type="/14"/>
<dotnet:Attribute name="birthdate" owningClass="/10" type="/18"/>
<dotnet:Type/>
<dotnet:Attribute name="firstname" owningClass="/10" type="/14"/>
<dotnet:Attribute name="lastname" owningClass="/10" type="/14"/>
<dotnet:Folder name="Login.ems.graphml" owningFolder="/3" files="/22 /56" namedElement="/72"/>
<dotnet:UserControl name="Login.ascx" owningFolder="/21" components="/55 /65 /66" codeBehind="/56"/>
<dotnet:WebPage name="welcome.aspx" owningFolder="/3" components="/22"/>
<dotnet:Folder name="OpenPoll.ems.graphml" owningFolder="/3" files="/25" namedElement="/75"/>
<dotnet:UserControl name="Open Poll.ascx" owningFolder="/24"/>
<dotnet:WebPage name="homepage.aspx" owningFolder="/3" components="/25"/>
<dotnet:Folder name="Voting.ems.graphml" owningFolder="/3" files="/28 /30" namedElement="/77"/>
<dotnet:UserControl name="Enter Voter.ascx" owningFolder="/27"/>
<dotnet:WebPage name="Vote.aspx" owningFolder="/3" components="/28 /30"/>
<dotnet:UserControl name="Show Ballot.ascx" owningFolder="/27"/>
<dotnet:Controller name="Login.ems.graphmlController" owningFolder="/4" operations="/32 /69"
useCase="/72"/>
<dotnet:Operation name="findUser" owningElement="/31" callee="/33" parameters="/64 /67 /68"/>
<dotnet:Operation name="findUser" owningElement="/6" callee="/37" parameters="/34"/>
<dotnet:Parameter name="User" type="/35" owningOperation="/33"/>
<dotnet:Type auxiliaryType="/5"/>
<dotnet:Parameter name="username" type="/14" owningOperation="/2"/>
<dotnet:Operation name="findUser" owningElement="/7" callee="/2" parameters="/38 /40"/>
<dotnet:Parameter name="username" type="/14" owningOperation="/37"/>
<dotnet:Parameter name="password" type="/14" owningOperation="/2"/>
<dotnet:Parameter name="password" type="/14" owningOperation="/37"/>
<dotnet:Controller name="Voting.ems.graphmlController" owningFolder="/4" operations="/42"
useCase="/77"/>
<dotnet:Operation name="findTerm" owningElement="/41" callee="/43"/>
<dotnet:Operation name="findTerm" owningElement="/11" callee="/47" parameters="/44"/>
<dotnet:Parameter name="Term" type="/45" owningOperation="/43"/>
<dotnet:Type auxiliaryType="/10"/>
<dotnet:Parameter name="lastname" type="/14" owningOperation="/9"/>
<dotnet:Operation name="findTerm" owningElement="/12" callee="/9" parameters="/48 /50 /52 /54"/>
<dotnet:Parameter name="lastname" type="/14" owningOperation="/47"/>
<dotnet:Parameter name="firstname" type="/14" owningOperation="/9"/>
<dotnet:Parameter name="firstname" type="/14" owningOperation="/47"/>
<dotnet:Parameter name="birthdate" type="/18" owningOperation="/9"/>
<dotnet:Parameter name="birthdate" type="/18" owningOperation="/47"/>
<dotnet:Parameter name="pin" type="/14" owningOperation="/9"/>
<dotnet:Parameter name="pin" type="/14" owningOperation="/47"/>
<dotnet:TextBox name="Username" owningComposite="/22"/>
<dotnet:CodeBehind name="Login.cs" owningFolder="/21" webForm="/22" operations="/57"/>
<dotnet:Operation name="Login_Click" owningElement="/56" callee="/69" parameters="/58 /61"/>
<dotnet:Parameter name="sender" type="/60" owningOperation="/57"/>

```

```

<dotnet:LibraryClass name="Object"/>
<dotnet:Type auxiliaryType="/59"/>
<dotnet:Parameter name="e" type="/63" owningOperation="/57"/>
<dotnet:LibraryClass name="EventArgs"/>
<dotnet:Type auxiliaryType="/62"/>
<dotnet:Parameter name="Username" type="/14" owningOperation="/32"/>
<dotnet:Button name="Login" owningComposite="/22" onClick="/57"/>
<dotnet:TextBox name="Password" owningComposite="/22" textMode="Password"/>
<dotnet:Parameter name="Password" type="/14" owningOperation="/32"/>
<dotnet:Parameter name="loggedUser" type="/35" owningOperation="/32"/>
<dotnet:Operation name="setLoggedUser" owningElement="/31" parameters="/70 /71"/>
<dotnet:Parameter name="foundUser" type="/35" owningOperation="/69"/>
<dotnet:Parameter name="loggedUser" type="/35" owningOperation="/69"/>
<dotnet:UseCase name="Login.ems.graphml" nextStep="/74" controller="/31" folder="/21"/>
<dotnet:UseCase name="welcome" controller="/78" folder="/79"/>
<dotnet:UseCase name="homepage" controller="/83" folder="/84"/>
<dotnet:UseCase name="OpenPoll.ems.graphml" nextStep="/76" controller="/82" folder="/24"/>
<dotnet:UseCase name="voting" controller="/80" folder="/81"/>
<dotnet:UseCase name="Voting.ems.graphml" nextStep="/73" controller="/41" folder="/27"/>
<dotnet:Controller name="welcomeController" owningFolder="/4" useCase="/73"/>
<dotnet:Folder name="welcome" owningFolder="/3" namedElement="/73"/>
<dotnet:Controller name="votingController" owningFolder="/4" useCase="/76"/>
<dotnet:Folder name="voting" owningFolder="/3" namedElement="/76"/>
<dotnet:Controller name="OpenPoll.ems.graphmlController" owningFolder="/4" useCase="/75"/>
<dotnet:Controller name="homepageController" owningFolder="/4" useCase="/74"/>
<dotnet:Folder name="homepage" owningFolder="/3" namedElement="/74"/>
</xmi:XMI>

```

D.5 andromDA

Listing 23 presents the EMF created by the apsm-to-andromda transformations for the use cases of the EMS in Chapter 9.

Listing 23 The EMF of the adromda of three use cases of the EMS

```

<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:andromda="http://andromda.modewis" xsi:schemaLocation="http://andromda.modewis
andromda.ecore">
  <andromda:Model name="Data" ownedMember="/1"/>
  <andromda:Package name="test" owner="/0" ownedMember="/2"/>
  <andromda:Package name="org.modewis.test" owner="/1" ownedMember="/3 /16 /23 /34 /35 /39 /43 /53 /65
/69"/>
  <andromda:Package name="Login.ems.graphml" owner="/2" ownedMember="/4 /8 /12"/>
  <andromda:UseCase name="Login.ems.graphml" behaviour="/5">

```

```

    <stereotypes>FrontEndUseCase</stereotypes>
</andromda:UseCase>
<andromda:StateMachine name="Login.ems.graphml" owner="/3" states="/6 /7 /29 /30 /31 /32 /33"
controller="/18"/>
    <andromda:State name="Error State 0" owner="/5">
        <stereotypes>FrontEndView</stereotypes>
    </andromda:State>
    <andromda:State name="Login" owner="/5">
        <stereotypes>FrontEndView</stereotypes>
    </andromda:State>
    <andromda:UseCase name="OpenPoll.ems.graphml" behaviour="/44">
        <stereotypes>FrontEndUseCase</stereotypes>
    </andromda:UseCase>
    <andromda:StateMachine name="OpenPoll.ems.graphml" owner="/3" states="/10 /11"/>
    <andromda:State name="Open Poll" owner="/9">
        <stereotypes>FrontEndView</stereotypes>
    </andromda:State>
    <andromda:State name="Error State 0" owner="/9">
        <stereotypes>FrontEndView</stereotypes>
    </andromda:State>
    <andromda:UseCase name="Voting.ems.graphml" behaviour="/54">
        <stereotypes>FrontEndUseCase</stereotypes>
    </andromda:UseCase>
    <andromda:StateMachine name="Voting.ems.graphml" owner="/3" states="/14 /15" controller="/21"/>
    <andromda:State name="Error State 2" owner="/13">
        <stereotypes>FrontEndView</stereotypes>
    </andromda:State>
    <andromda:State name="Error State 1" owner="/13">
        <stereotypes>FrontEndView</stereotypes>
    </andromda:State>
    <andromda:Package name="services" owner="/2" ownedMember="/17 /20"/>
    <andromda:Class name="UserService" owner="/16" operation="/26">
        <stereotypes>Service</stereotypes>
    </andromda:Class>
    <andromda:Class name="Login.ems.graphmlController" owner="/5" clientDependencies="/19"/>
    <andromda:Dependency supplier="/17" client="/18"/>
    <andromda:Class name="TermService" owner="/16" operation="/72">
        <stereotypes>Service</stereotypes>
    </andromda:Class>
    <andromda:Class name="Voting.ems.graphmlController" owner="/13" clientDependencies="/22"/>
    <andromda:Dependency supplier="/20" client="/21"/>
    <andromda:Package name="vo" owner="/2" ownedMember="/24 /70"/>
    <andromda:Class name="UserVO" owner="/23" attributes="/25">
        <stereotypes>ValueObject</stereotypes>
    </andromda:Class>
    <andromda:Attribute name="id" type="Long" owner="/24"/>

```

```

<andromda:Operation name="findUser" parameter="/27" owner="/17"/>
<andromda:Parameter name="userV0" type="/28" owner="/26"/>
<andromda:Type auxiliaryType="/24"/>
<andromda:State name="runState0" owner="/5"/>
<andromda:State name="choice0" owner="/5"/>
<andromda:State name="welcome" owner="/5"/>
<andromda:State name="homepage" owner="/5"/>
<andromda:State name="startState" owner="/5"/>
<andromda:Package name="domain" owner="/2"/>
<andromda:Package name="welcome" owner="/2" ownedMember="/37"/>
<andromda:StateMachine name="welcome" owner="/35" controller="/38"/>
<andromda:UseCase name="welcome" behaviour="/36">
  <stereotypes>FrontEndUseCase</stereotypes>
</andromda:UseCase>
<andromda:Class name="welcomeController" owner="/36"/>
<andromda:Package name="voting" owner="/2" ownedMember="/41"/>
<andromda:StateMachine name="voting" owner="/39" controller="/42"/>
<andromda:UseCase name="voting" behaviour="/40">
  <stereotypes>FrontEndUseCase</stereotypes>
</andromda:UseCase>
<andromda:Class name="votingController" owner="/40"/>
<andromda:Package name="OpenPoll.ems.graphml" owner="/2" ownedMember="/8"/>
<andromda:StateMachine name="OpenPoll.ems.graphml" owner="/43" states="/45 /46 /47 /48 /49 /50 /51"
controller="/52"/>
<andromda:State name="Open Poll" owner="/44"/>
<andromda:State name="runState0" owner="/44"/>
<andromda:State name="choice0" owner="/44"/>
<andromda:State name="Error State 0" owner="/44"/>
<andromda:State name="welcome" owner="/44"/>
<andromda:State name="startState" owner="/44"/>
<andromda:State name="voting" owner="/44"/>
<andromda:Class name="OpenPoll.ems.graphmlController" owner="/44"/>
<andromda:Package name="Voting.ems.graphml" owner="/2" ownedMember="/12"/>
<andromda:StateMachine name="Voting.ems.graphml" owner="/53" states="/55 /56 /57 /58 /59 /60 /61 /62
/63" controller="/64"/>
<andromda:State name="Show Ballot" owner="/54"/>
<andromda:State name="runState1" owner="/54"/>
<andromda:State name="choice1" owner="/54"/>
<andromda:State name="runState2" owner="/54"/>
<andromda:State name="choice2" owner="/54"/>
<andromda:State name="Error State 2" owner="/54"/>
<andromda:State name="welcome" owner="/54"/>
<andromda:State name="Enter Voter" owner="/54"/>
<andromda:State name="Error State 1" owner="/54"/>
<andromda:Class name="Voting.ems.graphmlController" owner="/54"/>
<andromda:Package name="homepage" owner="/2" ownedMember="/67"/>

```

```
<andromda:StateMachine name="homepage" owner="/65" controller="/68"/>
<andromda:UseCase name="homepage" behaviour="/66">
  <stereotypes>FrontEndUseCase</stereotypes>
</andromda:UseCase>
<andromda:Class name="homepageController" owner="/66"/>
<andromda:Package name="web" owner="/2"/>
<andromda:Class name="TermVO" owner="/23" attributes="/71">
  <stereotypes>ValueObject</stereotypes>
</andromda:Class>
<andromda:Attribute name="id" type="Long" owner="/70"/>
<andromda:Operation name="findTerm" parameter="/73" owner="/20"/>
<andromda:Parameter name="termVO" type="/74" owner="/72"/>
<andromda:Type auxiliaryType="/70"/>
</xmi:XMI>
```

Appendix E. GLOSSARY

This appendix lists the special terms used in this thesis and explains their definition throughout this thesis. While the definition of each term does not contradict with the general definition of that term in the area of software engineering, the definitions found in this appendix are special to this thesis.

Table 21. The Glossary

Term	Definition
Abstract Model	A model containing common features of several models while ignoring their differences.
Abstract Platform	A platform that addresses common techniques, technologies and patterns that exist within several platforms while ignoring their differences.
Abstract Web Platform	A model enclosing common features of other web models and technologies studied and reported in Chapter 3.
Abstract-Platform Specific Model	A model that is specific to the features contained in an abstract platform, which, in this thesis, refers to the abstract web platform detailed in Chapter 4.
APSM	When upper case: the meta-model presented in Chapter 4, which is the main contribution of this thesis. Also, see See Abstract-Platform Specific Model.
Apsm	When lower case: an instance of APSM.
Controller	A class that controls the behaviour of a use case. This class includes operations required to respond to user steps of the use case and direct them to proper services.
High-Level Input	Any visual model created using the language introduced in Section 4.8.

Term	Definition
Input	The source of any of the transformations implemented in this thesis but most of all <i>input</i> refers to any model that is used as the source of the pim-to-apsm transformations and adapts to the meta-model presented in Section 4.8.5.
Java Utility	Any of the Java applications developed to support this thesis by creating the <i>pim</i> from the input or by generating an application from a <i>spsm</i> .
Layer	A general concern in a conceptual or physical architectural model or pattern. This could be any of the CIM, PIM, PSM, APSM, or SPSM. It can also refer to any of the model-view-controller components of MVC. According to [Leth2001]: “A component is a layer if any calls to it come from more abstract components, and, if it calls anything, it calls more concrete components.”
Level	An indication of the degree of abstractness. The higher the level, the more abstract the model becomes. In this thesis, <i>pims</i> are considered higher-level, <i>apsm</i> is in a lower level and <i>spsms</i> are at the lowest level.
Mapping	An informal definition of how to relate instances of two meta-models. More specifically, a mapping defines how to obtain an instance of a meta-model, i.e. target meta-model from an instance of another meta-model, i.e. source meta-model.
Meta-Model	Any model that has the modeling elements required to define models that are the source and target of the transformations of this thesis. The most important meta-model is the APSM, which is the meta-model of the source/target models of the pim-to-apsm transformation as well as being the meta-model of the source model of any of the apsm-to-spsm transformations.

Term	Definition
Model	Typically an instance of a meta-model used in the context of a transformation.
Pim	When lower-case: an instance of a PIM.
PIM	When upper-case: The meta-model defined as a subset of the APSM defined in Section 4.8. It stands for Platform-Independent Model.
Platform	An integrated set of technologies, patterns, and functionality used to develop and deploy applications.
Service	A class that is responsible for preparing information requested by the controller. In this thesis, most services are data services that facilitate data-related operations.
Source Model	The model that is checked in a set of QVT relations. This model remains intact and may be compared to an input of a conventional computer program.
Specific Platform	A platform set up according to specific features and one or more concrete technologies.
Specific Web Platform	A specific platform designed for web-based applications. Specific web platforms in this thesis are introduced in Chapter 6.
SPSM	When in upper-case: any abstract model used to specify platform-specific models of applications based on the specific platforms defined in Chapter 5.
Spsm	When in lower-case: an instance of any SPSM.
Target Model	The model that is enforced in a set of QVT relations. This model is created according to the pattern defined in QVT relations and may be compared to an output of a conventional computer program.

Term	Definition
Transformation	A formal implementation of a mapping. Transformations in this thesis are sets of QVT relations and are presented in Chapter 5.