STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LI**, Number 2, 2006

ON MODEL-DRIVEN DEVELOPMENT FOR WEB APPLICATIONS

IOAN LAZĂR AND DAN COJOCAR

Abstract. The importance of requirements engineering for web systems is increasing today. Only few methodologies provides a systematic approach for the specification of web systems through requirements models. New results that address model transformation from requirements to web system design were recently obtained in the context of using QVT language.

In this paper we propose an approach for deriving web system design from requirements models in the context of a model-driven development process. We propose an extension of AndroMDA basic development process that can be suitable also for other model-driven processes. We also extend the AndroMDA presentation profile for modeling conversational flows for web applications.

1. Introduction

Current trends in software development focus on the specification of models and model transformations. Model-driven development is a successful methodology for model transformations based on Model Driven Architecture [8]. Model-driven development starts at the computational independent level (CIM) with a business model that capture system requirements. Then the initial CIM model is refined and a platform independent model (PIM) is obtained. Finally, the code is generated by transforming the PIM model into a platform specific model (PSM).

In this paper we focus on early steps of model-driven development: obtaining a PIM model from business requirements. Model transformations from requirements to web system designs were recently investigated by Koch et al [16]. In order to capture web process models they used activity diagrams and introduced an UML profile based on requirements metamodels introduced in [4]. Our approach is different because we focus on using *UML 2.0 state machines* to express the process models in web applications. In this paper we propose to use standard UML and

Received by the editors: November 20, 2006.

2000 *Mathematics Subject Classification.* 68U07, 68U35.

1998 *CR Categories and Descriptors.* D.2.1 [**Requirements/Specification**]: Methodologies; D.2.2 [**Design Tools and Techniques**]: Computer-aided software engineering (CASE), Object-oriented design methods, State diagrams; D.2.11 [**Software Architectures**]: Domain-specific architectures.

the extension mechanism provided through stereotypes [10]. The intent is to allow the modeling process to be performed using any modeling tool that conforms to UML 2.0 and UML extension mechanism.

State machine models were also used by AndroMDA organization[1] which developed several profiles and transformation modules called cartridges. AndroMDA provides also transformation cartridges for Struts [5] (see [2]) and Java Server Faces. We also introduce a new profile that extends AndroMDA profile for presentation layer. This profile allows to model conversational flows in web applications. Moreover the profile is designed such that to allow transformations for other frameworks, such as Spring MVC [1] and Spring WebFlow [3].

The remainder of this paper is structured as follows. Section 2 introduces a new model-driven development process for web applications. A new UML profile for presentation layer is defined in Section 3. Section 4 discusses how the profile and the model-driven development process can be used together. Finally, in Section 5 some conclusions and future work are outlined.

2. Model-Driven Development for Web Applications

The basic idea of model-driven development is to build platform independent designs and to generate code for specific platforms. Recently Koch et al [16, 4] introduced a model-driven approach for web systems. Their approach are structured around OO-H method [7, 14] and UWE [12, 11, 13, 15]. The PIM models used by the authors are:

- *Process model* – defines business processes/workflows using UML activities. This is the main model from which the content and navigation models can be derived using Query/View/Transformation [9].
- *Content model* – contains objects needed for the construction of web pages content. The classes of this model can be generated from the process model.
- *Navigation model* – defines page flow navigation and menus organization. Also this model can be generated from the process model using QVT transformations.
- *Presentation model* – represents pages layout and design.

The basic idea in [16, 4] is to use activity diagrams for describing web business processes and then to derive other PIM artifacts using QVT transformation rules. Our approach is to use state machine diagrams instead of activity diagrams. UML state machine semantic is more suitable for modeling web user interface than UML activity diagrams. Other differences between our approach and the approach from [16, 4] are: (a) the *Content model* will be derived from signal properties associated with transitions between states; (b) our approach is suitable for processes that produces web systems having the modern layer architecture presented in Figure 1.

[1]http://www.andromda.org/

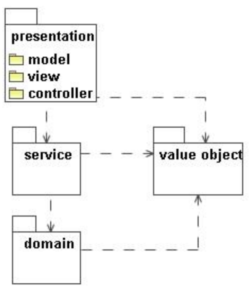


FIGURE 1. Web Application Architecture

The *presentation* package from Figure 1 corresponds to the presentation layer and includes web pages, required form models, and controllers. The *service* package corresponds to the business layer and contains business services. The *domain* package will include entities and business objects. The *value object* package includes coarse grained objects used to transfer data between *domain* and *presentation.* As Figure 1 shows, *presentation* will use only *services* and *value objects.*

In this paper we refine the basic MDA process steps presented in Figure 2 (A). This process was introduced by AndroMDA organization as its basic development approach. It is a model-driven, agile, test-driven, and iterative development process. Each iteration starts with *modeling the current iteration scenarios* (as PIM) – agile and iterative approach. Then automatic generation of *entities and services* (PSM) follows – model-driven approach. Then, *writing unit tests* for services (before implementing the service logic) – test-driven approach, *implementing the service logic*, and *running units tests* follows. *Implementing the front-end* is the last step in this development process.

The focus of this paper is on web requirements engineering, that is on the first step of the model-driven process from Figure 2 (A) – *model current iteration scenarios.* This step indicates what is needed to be build for the current iteration. Emphasizes on this step was also considered in [16]. Figure 2 (B) presents our extension to the basic model-driven approach from Figure 2 (A). Figure 2 (B) shows an activity diagram that refines the first step from Figure 2 (A).

Short descriptions of these tasks and their relationships with OpenUP [6] disciplines are given below:

- *Model use cases.* The primary focus is to obtain a detailed *use case model* for the current iteration scenarios. This step belongs to the OpenUP

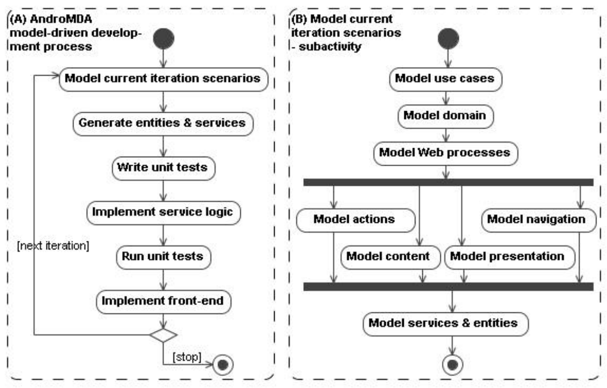


FIGURE 2. A model-driven development process

requirements discipline. The remaining steps that follows below belongs to the OpenUP analysis and design discipline.

- *Model domain.* In this step we identify the elements (classes, subsystems, etc) that collaborate together to provide the required behavior. The artifact produced after this step is a *conceptual/domain model.*
- *Model web processes.* In this step we determine how elements collaborate to realize the scenarios at high level. In order to do that we define web user interface flows by attaching *state machine diagrams* to use cases. This is an analysis step and not a design step. So, at this stage the state machines do not capture *actions* needed to be performed in order to obtain information from the system. The state diagrams express only *views* and *flows* between them.
- *Refine web processes.* The scope of this step is to detail how elements collaborate to realize the scenarios – that is, a design step. Major design decisions are made in this stage. Ideally the following steps are performed, sequentially:
  - *Model actions.* For each use case a *controller* class is attached. Now the state machines are refined by adding *action states* that captures the required system operations to be performed. The action states

will defer execution to controller operations. The controller operations will be manually implemented by developers.

  – *Model content. Presentation form model* must be discovered at this step, that is what data need to be presented in *view states.* In order to keep things simple and independent of specific frameworks we follow the basic idea from [2], that is, we do not model explicitly presentation form models. Instead we model data propagation between *action states* and *view states* within our state machine diagrams. At this stage we introduce *value objects* that will carry data from *domain* to *presentation.* Data propagation between action and view states are modeled as parameters assigned to the transition events. PIM to PSM transformation processes need to generate the required presentation form models from data propagation between states.
  – *Model navigation.* Web pages navigation will be directly generated from transitions between states. Special considerations require global transitions and menus.
  – *Model presentation.* This refinement step refers to marking event parameters sent between *action* and *view states* so that the generated code includes the required page layout and controls.

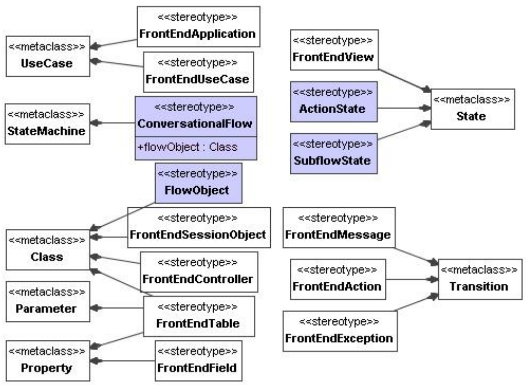
- *Model services and entities.* The connection between *presentation* layer and *service* layer is designed at this stage. The required services are designed and the relationships between each controller and required services are established. Moreover, the refinement of domain entities occurs also.

3. A New Presentation Profile for Conversational Flows

We extend AndroMDA profiles for modeling web applications [2] by adding the concept of a *conversational* scope to support the execution of use cases that span multiple use cases [17, Chapter 11]. Many web applications have use cases that do not normally fit into the request, session, and application scopes defined by the Java servlet specification. Such applications have use cases that span more than one page but do not require the longevity of the session.

Spring Web Flow framework [3] treats conversational flows as first level citizens. In this section we introduce new UML stereotypes for modeling conversational flows in UML. The UML models marked with these stereotypes will allow transformation tools to extract flow information for specific target platforms like [3]. These new stereotypes are presented below.

ConversationalFlow: This stereotype can be applied on state machines and indicates that the contained front end view states represents a single user conversation. During this user conversation the front end views can



<<metaclass>> UseCase
<<stereotype>> FrontEndApplication
<<stereotype>> FrontEndUseCase
<<metaclass>> StateMachine
<<stereotype>> ConversationalFlow
+flowObject : Class
<<stereotype>> FlowObject
<<stereotype>> FrontEndSessionObject
<<metaclass>> Class
<<stereotype>> FrontEndController
<<metaclass>> Parameter
<<stereotype>> FrontEndTable
<<metaclass>> Property
<<stereotype>> FrontEndField
<<stereotype>> FrontEndView
<<stereotype>> ActionState
<<metaclass>> State
<<stereotype>> SubflowState
<<stereotype>> FrontEndMessage
<<stereotype>> FrontEndAction
<<metaclass>> Transition
<<stereotype>> FrontEndException

FIGURE 3. A new profile for conversational flow

share some information. The *flowObject* tag can be used to indicate a class that encapsulate the conversation shared information.

FlowObject: This stereotype is used to mark classes that represents information used in conversational flows. When transforming the PIM model into PSM models the dependency relationships between flow object classes and front end controllers classes can be used to generate convenient methods to access flow object properties within controller classes.

ActionState: This stereotype can be used to mark server side actions that belongs to conversational flows. AndroMDA profiles do not provide a stereotype for server side actions, by default a state not marked with *FrontEndView* stereotype being considered a server side action state.

SubflowState: The submachine states within a *ConversationalFlow* state machine that span a subflow that is part of the conversation will be marked using *SubflowState*.

The other stereotypes presented in Figure 3 comes from AndroMDA profiles – *ValueObject* stereotype being part of the common profile and the remaining stereotypes being part of the presentation profile. The next section refers almost all these stereotypes.

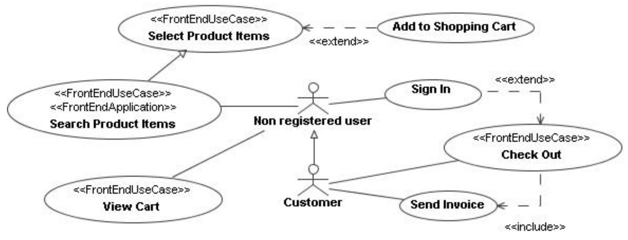


Figure 4. Partial use case diagram of Amazon web application

4. Modeling Example

In this section we outline the steps presented in Figure 2 (B) on Amazon[2] running example (also used as case study in [13]). The following actors interact with the application: (a) non registered user – search and select products; add products to the shopping cart, and login (b) customer - inherits from the non registered user and it is allowed (after logged-in) to start the checkout proces.

The subsections of this section correspond to the tasks presented in Figure 2 (B). Content, navigation, and presentation modeling tasks will not be discussed below, these tasks being well documented in [2]. We consider that all requirements outlined above are allocated for the iteration described in the subsections that follow. Each task includes several marking steps that indicate how to build the model in order to transform later the model into a PSM model.

4.1. **Model use cases task.** The purpose of this task is to detail the current iteration requirements, and the result is a use case model. This task includes the following marking steps [2]:

Step 1: Mark with *FrontEndView* stereotype those use cases that require user interaction.

Step 2: Mark with *FrontEndApplication* a single use case that will represent the application entry point.

Figure 4 presents a use case model for our example and the given requirements above. *Search product items*, *View cart*, and *Check out* use cases are marked with *FrontEndView*, and *Search product items* is also marked with *FrontEndApplication* stereotype.

[2]http://www.amazon.com

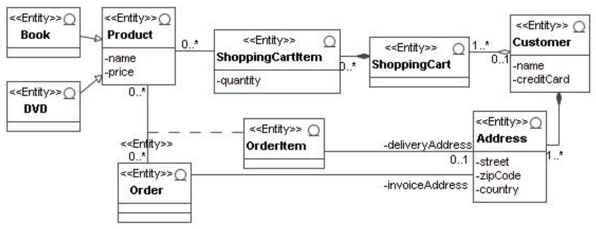


FIGURE 5. Domain model

4.2. **Model domain task.** The primary purpose of this task is to identify the concepts used to provide the required behavior. At this stage we produce a conceptual model presented in Figure 5. This is a common feature in most web modeling approaches including UWE and OO-H. The marking step of this task is [2]:

Step 3: Apply the *Entity* stereotype to all domain persistent objects.

When transforming the PIM model, the UML entities will be mapped to entities of specific object relational mapping frameworks.

4.3. **Model web processes.** The primary purpose of this task is to describe at high level the web business processes using UML 2.0 state machines. At this level of abstraction the state machines will include only the view state. The marking steps of this task are:

Step 4: For each *FrontEndUseCase* attach a state machine that describes the required views (web pages) and the navigation between them.
Step 5: Use UML submachine states in order to model navigation between use cases.
Step 6: Mark state machines with *ConversationalFlow* stereotype where appropriate. Mark where appropriate the submachine states of a *ConversationalFlow* with *SubflowState* stereotype.

This task is not indicated in [2]. In fact this is an intermediary step, that is, a requirements analysis step. The resulting state machine diagrams will be refined during the next design task.

Figure 6 and 7 presents the result of applying the steps 4–6 described above. *Checkout* web process is defined as conversational flow, that is all contained views will share some information (e.g. payment information is used when the invoice is submitted). Note different types of navigation between *Search product items* web process and: (a) *Show cart* state machine attached to the use case View cart, and (b) *Checkout* conversational flow.

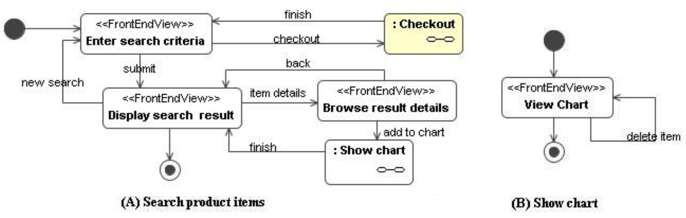


FIGURE 6. *Search product items* and *View cart* web processes

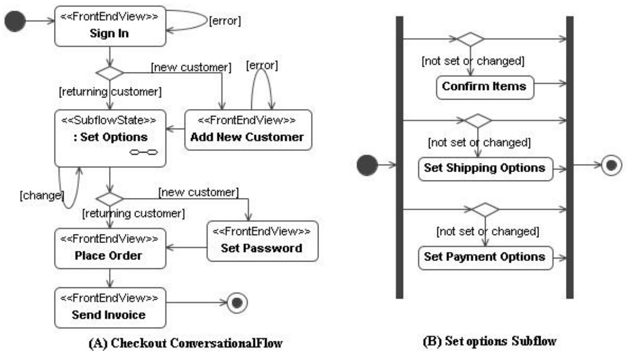


FIGURE 7. *Checkout* web process

Instead on relying on submachines for navigation, the reference documentation of AndroMDA for Struts [2] indicates the usage of labels attached to the ingoing transitions of final states.

4.4. **Refine Web processes.**

Model actions task. The purpose of this task is to determine the system operations. In order to do that we refine the web processes by adding action states corresponding to server-side action needed to be executed. Because current server-side frameworks do not follow a standard defined architecture we follow the guideline from [2] and the actions will be encapsulated in front-end controllers. The relationships between front-end controllers and session/conversational flow objects will be generated according to the targetted platform.

Step 7: For each *FrontEndUseCase* attach a controller through a tag defined by this stereotype.

Step 8: Optionally, mark each state that represents server-side actions with *ActionState* stereotype.

Step 9: Indicate the actions that need to be executed when entering action state using deferrable trigger property of a UML state. The triggers should refer a UML CallEvent pointing to a controller operation.

Step 10: For each transition between action and view states use signal parameters to indicate the data flow between those states. At this step usually we introduce classes that captures data extracts from domain – those classes will be marked with *ValueObject* stereotype.

Step 11: Define session objects and mark their classes with *FrontEndSession*. Add dependency relationships between controllers and session object classes.

Step 12: Define objects with conversational scope and link them with conversational state machine using the tag *flowObject* (see Figure 3).

The steps 7–11 are standard steps indicated in [2]. Only the last step is new and this is needed only for the new introduced conversational flows.

5. Conclusions

The extension of AndroMDA model-driven development process with emphasizes on requirements engineering provides a simple approach close to traditional requirements engineering for desktop applications. The main advantage of using such a process consist in the ability to perform requirements engineering tasks at a high level of abstraction, independent of specific platform.

Currently we are analyzing the possibilities to extend AndroMDA profile in order to support full mapping to other frameworks such as SpringMVC, and Spring Web Flow.

References

[1] ***. Spring framework reference documentation. Technical report, 2006. http://static.springframework.org/spring/docs/2.0.x/spring-reference.pdf (06/07/06).

[2] AndroMDA. *Business Process Management for Struts Cartridge.* 2006. http://galaxy.andromda.org/docs/andromda-bpm4struts-cartridge/index.html.

[3] Keith Donald and Ervin Vervaet. *Spring WebFlow Reference Documentation.* 2006. http://static.springframework.org/spring-webflow/docs/1.0.x/spring-webflow-reference.pdf (06/07/06).

[4] María Josè Escalona and Nora Koch. Metamodeling the requirements of web systems. In *Proc. of the 2nd Int. Conf. on Web Information Systems and Technologies*, Setubal, Portugal, April 2006.

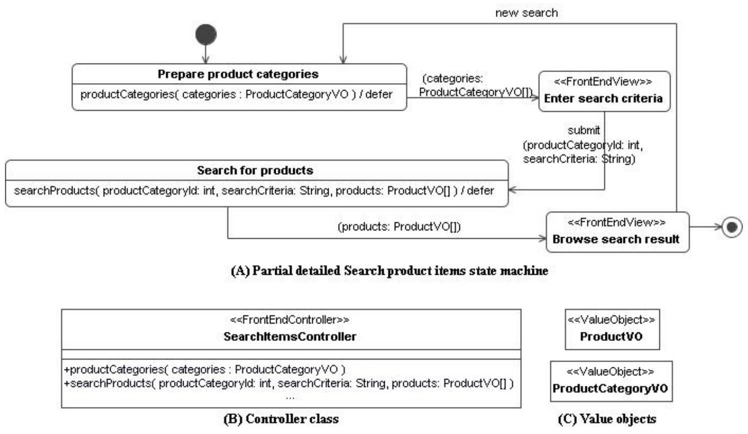

new search
Prepare product categories
productCategories(categories : ProductCategoryVO) / defer
(categories: ProductCategoryVO[])
<<FrontEndView>>
Enter search criteria
submit (productCategoryId: int, searchCriteria: String)
Search for products
searchProducts(productCategoryId: int, searchCriteria: String, products: ProductVO[]) / defer
(products: ProductVO[])
<<FrontEndView>>
Browse search result
(A) Partial detailed Search product items state machine
<<FrontEndController>>
SearchItemsController
+productCategories(categories : ProductCategoryVO)
+searchProducts(productCategoryId: int, searchCriteria: String, products: ProductVO[])
...
(B) Controller class
<<ValueObject>>
ProductVO
<<ValueObject>>
ProductCategoryVO
(C) Value objects

FIGURE 8. Detailed web process - action and view states

[5] Apache Foundation. *Struts Framework, Version 1.3.5*. 2006. http://struts.apache.org/1.3.5/index.html.

[6] Eclipse Process Framework. *OpenUP/Basic*. 2006. http://www.eclipse.org/epf/ (06/07/06).

[7] Jaime Gomez and Cristina Cachero. Oo-h method: Extending uml to model web interfaces. In *Information Modeling for Internet Applicaions*, pages 144–173. Idea Group Publishing, 2002.

[8] Object Management Group. *MDA Guide Version 1.0.1*. 2003. http://www.omg.org/docs/omg/03-06-01.pdf (06/07/06).

[9] Object Management Group. *MOF 2.0 Query/Views/Transformations RFP*. 2004. http://www.omg.org/cgi-bin/apps/doc?ad/02-04-10.pdf (06/07/06).

[10] Object Management Group. *UML 2.0 Superstructure*. 2004. http://www.omg.org/cgi-bin/apps/doc?formal/05-07-04.pdf (06/07/06).

[11] Alexander Knapp, Nora Koch, and Gefei Zhang. Modeling the behavior of web applications with argouwe. In *Lecture Notes in Computer Science*, pages 624–626. Springer Verlag, 2005.

[12] Nora Koch and Andreas Kraus. The expressive power of uml-based web engineering. In *Second Int. Workshop on Web-oriented Software Technology*, 2002.

[13] Nora Koch and Andreas Kraus. Integration of business processes in web application models. *Journal of Web Engineering*, 1(1):22–49, 2002.

[14] Nora Koch and Andreas Kraus. Modeling web business processes with oo-h and uwe. In *Third Int. Workshop on Web Oiented Software Technology*, 2003.

[15] Nora Koch and Andreas Kraus. Towards a common metamodel for the development of web applications. In *Second Int. Conference on Web Engineering*, pages 497–506. Springer Verlag, 2003.

[16] Nora Koch, Gefei Zhang, and Mar'ia Jos'e Escalona. Model transformations from requirements to web system design. In *Proc. of the 6th Int. Conf. on Web Engineering*, pages 281–288. ACM Press, 2006.

[17] Seth Ladd and Keith Donald. *Expert Spring MVC and Web Flows*. Apress, 2006.

E-mail address: `ilazar@cs.ubbcluj.ro`

E-mail address: `dan@cs.ubbcluj.ro`

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, 400084 M. Kogalniceanu 1, Cluj-Napoca, Romania