Proceedings of the 7th International Conference on Applied Informatics
Eger, Hungary, January 28–31, 2007. Vol. 2. pp. 173–179.

Combining the benefits of MVC design pattern and UML based modeling for different software platforms

Attila Adamkó, Csaba Bornemissza

Department of Information Technology, University of Debrecen
e-mail: adamkoa, bornem@inf.unideb.hu

Abstract

The Model View Controller design pattern is a good approach to keep our source code in a well structured form. Using UML modeling tools to design our software makes it possible for us to generate all these important components from the UML class diagram(s): Entity-Relationship model, SQL scheme, EJB Session Façade, XML Web Services, skeleton source code for the business logic.

In our article we present a combination of technologies which we find very effective for complex software development processes. We will show how to build a 3-tier application which is mainly generated from a set of simple UML class diagrams. After the design process the generated code is separated to the appropriate tiers of the application. The MVC pattern brings structure to the client side code. We will discuss the benefits and disadvantages of this strongly model-driven software development approach.

Categories and Subject Descriptors: D.2.10 [Software Engineering]: Design; D.2.11 [Software Engineering]: Software Architectures; H.4.3 [Information Systems Applications] Communications Applications

Keywords: MDA, Model View Controller, Enterprise Java Beans, UML Modeling, Hibernate

1. Introduction

The difficulty of developing Web applications increases as the number of technologies they use increases and as the interactions between these technologies become more complex. With the evolution of technologies the early static web sites are changing into Web based distributed applications. However, the reorganization and the development require knowledge and integration of several different technologies.

Moreover, an important factor in the development for such Web applications is the support of successful communication - using a common language - to avoid misunderstandings and expensive redesign. The UML (Unified Modelling Language) fulfils these requirements by providing a family of intuitive notations and diagrams which are could be used to describe software systems at a high level of abstraction. These models have to be focused on the information relevant to the different roles in the development team. These models make it possible for the team members to investigate the system of different aspects.

Performing early demonstrations is another important factor in the course of the communication with the customer. Therefore, the methodologies should support code-generation, which could shorten production time too. The models should formulate the complete structural description of the website, resulting a working, but incomplete prototype.

However, most of the currently available development tools are based on the J2EE specification for enterprise applications where Web applications have three or four tier architecture. We will show how to build a 3-tier application which is mainly generated from a set of simple UML class diagrams. After the design process the generated code is separated to the appropriate tiers of the application. The MVC pattern brings structure to the client side code. We will discuss the benefits and disadvantages of this strongly model-driven software development approach. We will highlight the benefits of the presented approach which helps in the development of medium- and big sized Web applications using UML, Web Services and Enterprise Java Beans.

2. Model Driven Architecture

The Model Driven Architecture (MDA) development lifecycle is very similar to the traditional lifecycle. The same phases are identified, but the main difference lies in the nature of the artifacts that are created during the development phases. The artifacts are formal models providing a given aspect of the system.

The first MDA model is a model with high level of abstraction that is independent from any implementation detail. This is called a Platform Independent Model (PIM). The PIM models are giving viewpoints of how the system will support the business. These models do not take care of implementation details like relational databases, application servers and so on. Just concentrate on the formalization of the requirements. Naturally, when we speak about models, we could make a distinction between business and software models or in another aspect, between structural and dynamic models. A system being modeled contains both aspects. In UML the relationship between dynamic and static views is direct, because they show different visualization of the same thing in the same model – like state diagrams and class diagrams.

The next step is the transformation of these models into one or more Platform Specific Models (PSM-s). The transformation process will deal with the available implementation technologies. For each specific technology a separate PSM is gen-

erated because most of the systems today span several technologies. In our case for example an EJB PSM is a model of the system in the terms of EJB structures like "entity bean" or "session bean" or another PSM will hold the database specific structures with the help of Hibernate classes.

The final step in the development process is the transformation of each PSM to code. Because a PSM fits its technology rather closely this transformation could be done relatively easy. Naturally, we could create our own transformation code too.

The MDA also defines how these models relate to each other. The most complex step is the transformation of the PIM into one or more PSM-s. We need to find the proper mapping between these models. Moreover, using these models offers a great benefit over the traditional software development processes. While original design and implementation can split over time during the traditional development, utilizing the MDA development life cycle we could avoid this by the automated transformation processes.

3. The generation process

In this paper we would utilize OMG's XML Metadata Interchange (XMI) format as an intermediate format that serves as a basis of the generator tools.

In our experimental project we used UML class diagrams for modeling the structure of the application. It is important to mention, that besides UML class diagrams there are other types of diagrams that can be used as a basis of code generation. Activity diagrams are useful to model the workflow of the planned application. The connections of the different activities define the navigation among the different forms of the user interface.

Class diagrams are useful to create model which describes several aspects of the planned software. Using stereotypes we distinguish classes with different roles in the system. Different class stereotypes are processed with different code generator definition sets. This creates an environment, where the classes and associations on a class diagram are generated into separated and inhomogeneous sets of code. We shall cover the most important stereotypes used in our model.

- **Entity.** Classes defined with the Entity stereotype will represent an entity in the entity-relationship model. Associations between entities represent foreign key constraints. The output generated from Entity classes is HBM XML descriptors which define an entity for the Hibernate persistence layer. From these XML files the Hibernate system will generate further Java classes that represent the E/R entities in the Java code.

- **Value Object.** VO classes are used as communication objects that transport information between the client and the server. VO-s are also used for internal processing tasks. VO-s are Plain Old Java Objects (POJO) with simple structure. Apart from constructor logic, these classes contain no code at all.

- **Service.** Service classes contain the server side methods, that are published for client side access. These classes will appear in the system as Stateless Session beans. The needed interfaces for home and remote methods will be generated automatically as well. Later on the client side stub files will be generated from these EJB sources for the XML Web Service accessibility. EJB interfaces and implementation classes will be stored at different locations, thus providing an environment, where the programmers know, which classes are free to modify, and which are strictly generated sources

- **Enumeration.** Enumerations are needed generally en every application development, these are supported by the code generator.

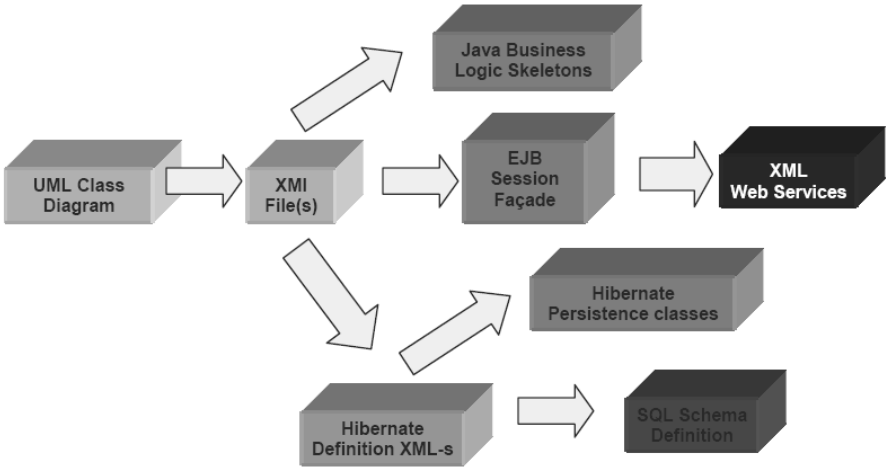

Figure 1: A possible automated code generation process

4. The system architecture

Figure 2 demonstrates the system architecture. A considerable amount of source will be generated code. It is important to know, which are the parts that are only generated, and where are the human-created program logics. We shall discuss this question for the different tiers of the application.

Database Layer. The schema definition part of the database is strictly generated code. The Hibernate compatibility would fail, if these definitions were altered by the developer team. However, there are functionalities in a database that are not automatically generated with a schema definition. Triggers, Stored procedures, other intelligent components like views, functions are created by the developer team. These sources must follow the schema model changes. Stored procedure, trigger, etc., creation scripts will be run after the schema creation scripts at deployment, thus incompatible changes can be identified in early testing phases.

Hibernate Persistence Layer. The HBM definition files are automatically generated. Any extra properties which affect the hibernate classes should come from the UML model. No manual change is allowed in these definition files. The Java classes that represent the Hibernate entities are generated, and no implementation logic is needed. There are custom Hibernate query possibilities, even using the stored procedures which provide flexibility for optimized data access. These solutions can be implemented without altering the Hibernate entity classes.

Business Logic Layer. Only human created source code belongs here. It is encouraged to create custom business classes, which are called and used from the EJB session bean methods.

EJB Session Façade. This is the application layer, where most of the human made source code appears. The generated source part means the Enterprise Java Bean interfaces and implementation classes, enumerations and value objects. The EJB implementation classes contain the business logic methods that are available from the client side. All server-side validations take place here. Besides that it is important to mention the annotation code, which will instruct the WebServiceGen tool, that creates the sources needed to provide web service level access to the Session Bean methods.

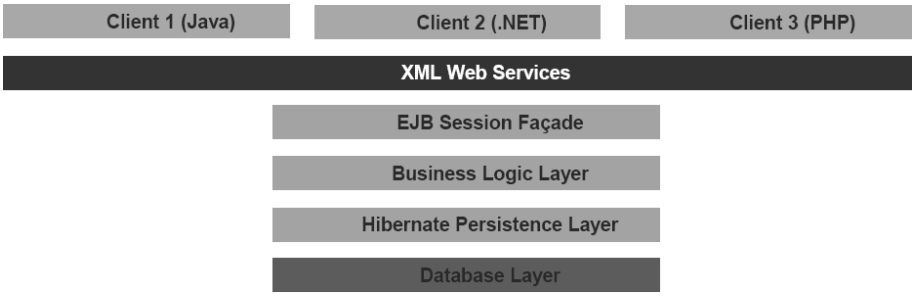

Figure 2: The system architecture

5. The Client side and the MVC pattern

We know that several similar problems could occur over and over again in the development process and we know that the design patterns could help to reuse successful design and architectural solutions. In the design of Web applications a useful way is indicated by the Model-View-Controller (MVC) design pattern which is adopted in several frameworks. The MVC pattern is very simple, yet incredible useful.

In the client side its importance lies in the clear separation of the functional layers and their functionality enforcing structured and organized code. Clearly separates the presentation aspects from the business logic using views and controller units together with model. Moreover, the Controller part could provide general entry points for the activity logging subsystem or for the authentication system.

We used GUI prototyping as a user interface design step. Creating a prototype gives the customer feedback possibility. Using a navigable HTML prototype is understandable for a customer without special skills in information technologies. Only customers with IT expertise would be able to process and reply a UML diagram based prototype.

6. Conclusion

One of the greatest advantages of the Model Driven Architecture is that we can build complex systems with a clear, structured project environment. The project participants (analysts, designers, developers) work on separated areas. It is well defined, on which files should a certain project member work, and which ones are read-only. The developers get generated implementation files, where they can focus on the clear implementation tasks.

Because of the well defined nature implementation tasks, a less experienced developer team can realize most of the tasks with the help of an experienced project architect.

Using Enterprise Javabeans and Hibernate increases the scalability of the system. Adding extra hardware, or separating different system layers to different hardware is a configuration issue, not an implementation issue.

Choosing XML Web Services as client-server communication protocol provides an industry standard accessibility to the system. If the need of external interface publishing occurs, the system is extendable, without any architecture changes.

The physical design plans are UML documents. These documents change along with the development process. Thus the design plans and the implementation code will strictly keep together during the project lifecycle. In older approaches it is very common that the implementation differs from the design documentation.

We must mention the possible **disadvantages** of this architecture as well.

Because of the big amount of generated sources, the system is not quite flexible against changes in the software functionality. Modifications made in the Model cause that the implementation logic will not compile, or will not function. This approach is not recommended for agile project with high change request expectations.

This “heavyweight” architecture is not recommended for smaller application developments either. The setup of the configuration and the whole generation process would only mean an overhead cost in the development.

As **conclusions**, we find it important to underline that this software development method is effective in medium and big size projects. Using agile project management methods along with this Model Driven Architecture is not recommended, especially when the project iteration times are shorter.

In bigger size development projects, with longer iteration periods, this development architecture creates a clean, effective project environment.

References

[1] Iverson, K., Real World Web Services, (2004).

[2] Topley, K., Java Web Services in a Nutshell, (2003).

[3] Snell, J., Tidwell, D., Kulchenko, P., Programming Web Services with SOAP.

[4] Elliott, J., Getting Started with Hibernate 3, (2006).

[5] Monson-Haefel, R., Burke, B., Enterprise JavaBeans 3.0, Fifth Edition, (2006).

[6] W3C - World Wide Web Consortium, `http://www.w3.org/`

[7] Object Management Group, The: XML Metadata Interchange (XMI), OMG document, (2001).

[8] Schattkowsky, T., Lohmann, M., Rapid development of modular dynamic web sites using UML, *In Proc. of 5th International Conference on UML 2002*, Springer, LNCS 2640, (Oct. 2002), 336–350.

Attila Adamkó, Csaba Bornemissza
Department of Information Technology
University of Debrecen
H-4010, P.O. Box 12, Debrecen
Hungary