

Stable Model-View-Mapping (MVM) Architectural Pattern

Mohamed E.Fayad¹, Nayeem Islam², and Haitham Hamza³

¹*Computer Engineering Department*

San José State University

San José, CA 95192-0180

Ph: +1 408 924-7364, Fax: +1 408 924-4153

m.fayad@sjsu.edu

²*DoCoMo Communication Laboratories*

San Jose, CA 94303

nayeem@docomolabs-usa.com

³*Computer Science and Engineering Department*

University of Nebraska-Lincoln

Lincoln, NE 68588, USA

Ph: +1 402 4729492

hhamza@cse.unl.edu

ABSTRACT

*Before heterogeneous devices will be able to access the same application, a mechanism for generating multiple views within a single model is needed. Current approaches for generating multiple views within a single model are greatly limited. By applying software stability concepts [10], this paper is able to propose a high-level architecture pattern, **Stable Model-View-Mapping (MVM)**, as a novel approach for generating multiple views from multiple models and vice versa. The proposed architecture confronts the limitations of current approaches and considers the mapping when designing the solution.*

1. INTRODUCTION

The ability of heterogeneous devices to access the same application imposes great complications in the design of these applications. Successful applications will be accessible from several devices. This demanding issue has sparked the need to develop mechanisms by which an abstract model can be mapped into different abstract views and vice versa. In addition to the issue of mapping between views and models, the current resources and capabilities of mobile and wireless devices raises other critical concerns.

Software stability concepts introduced in [10] have demonstrated great promise in the area of software reuse. Software Stability Models (SSMs) apply the concepts of Enduring Business Themes (EBTs) [8, 9, and 10] and Business Objects (BOs) [9, 10, 11, 12]. The stability and reusability properties of the EBTs and BOs qualify them to serve as a base for building stable and reusable patterns. In SSM the model of the system is viewed as three layers: the Enduring Business Themes (EBTs) layer, the Business Objects (BOs) layer, and the Industrial Objects (IOs) layer. Each class in the system model is classified into one of these three layers according to its nature. The properties that characterize EBTs, BOs, and IOs are given in [9]. Complete examples of building different systems using these software stability concepts can be found in [3,6,12].

Using SSM concepts, this paper proposes a high-level architecture pattern *Stable MVM* as a novel approach for generating multiple views from multiple models and vice versa. The proposed architecture confronts the limitations of current approaches by providing a flexible mapping between any models to any views and vice versa. In addition, this mapping can be carried over any media including the mobile environment while still considering the mobile media's resources and capabilities.

2. STABLE MODEL-VIEW-MAPPING PATTERN DESCRIPTION

Problem

How to build a high level architecture model that can provide, for any application, flexible mapping between any abstract model (which could either be a passive model, or a model returned by a specific application) to any abstract views and vice versa

Forces

- The problem of mapping between views and models spans a large spectrum of applications. Each application has different features and characteristics; therefore, developing a solution to serve as a base solution that handles the basic common issues between these applications is challenging.
- The pattern should deal with the situation of having multiple models within the applications. Current solutions assume only a single model.
- In many cases it is required to construct a view by composing multiple models within the applications.
- Sometimes the required view is constructed from the isolation of different parts from several models within the application. The pattern should be general enough to handle such situations.
- The pattern should encounter the possibility of generating models from different views (which is the reverse of the conventional functionality of the model-view mapping).
- The pattern needs to be abstract and not tied to any specific technology.

Pattern structure and Participants

Figure 1 shows the object diagram of the Stable MVM pattern. The shown model gives the high abstract level of the view for the proposed model.

The participants of the *Stable MVM* pattern are:

Classes:

- *Applicability*. Describes the application and the purpose for which mapping is needed. For instance, in one application a simple view that extract part of the data from the original model is needed for the seek of *simplicity*.

Patterns

- *AnyModel*. Describes the models within the application. The model is a representation of the data within the application.
- *AnyView*. Represents the view of a collection of data (the model).

- *AnyParty*. Represents both the modeler and the viewer. The modeler is responsible for building the data models in the appropriate abstract level. The viewer requests the model and the mapped view of that model.
- *AnyMedia*. Identifies and defines the media upon which the models and views are mapped and transmitted. It also represents the media by which the views are to be displayed (devices, PCs, etc.)
- *Mapping*. Defines the mapping rules between the models and their views. It also determines how this mapping will be performed.
- *Searching*. Searches AnyMedia for the requested application, model, or view.
- *AnyApplication*. Represents the application that is requested by AnyParty.

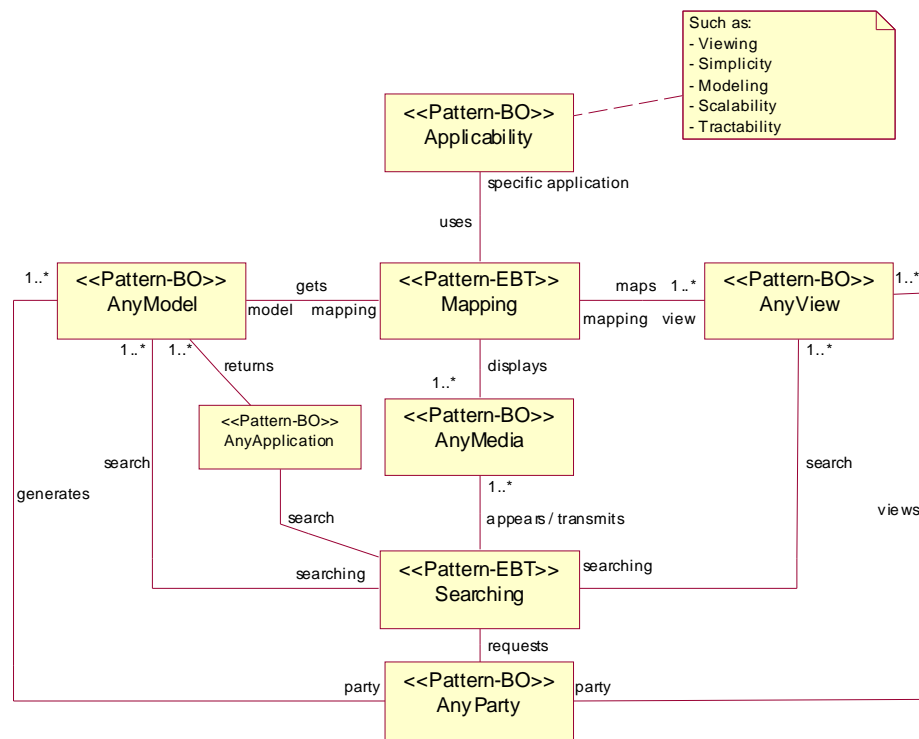


Figure 1. Stable MVM pattern object diagram

Implementation Issues

1. Separation of concerns:

Software stability separates the pattern into three horizontal abstraction layers (global view): EBTs, BOs, and IOs that can be dealt with individually with explicit dependencies among them. Each pattern is separately modeled into three horizontal abstraction layers underneath the global view (hidden view). Both layer of horizontal layers are vertical abstraction layers

2. Hooks:

Hooks are extension points on the BOs that can be adapted by AnyParty (e.g. application developers, users, operators). Each hook provides a specific requirement to be fulfilled

by AnyParty and documents how to extend the pattern or the framework to meet the requirements [14]. All BOs provide implemented hooks that can be utilized by IOs, application classes. For example AnyParty's hooks are the roles of all the users of the *Stable MVM* pattern, such as operator, user, technician, etc. You also can add, modify, activate, or disable many of the hooks on the fly, during run-time. For instance, Role-Object Pattern [15] allows for generating many roles /objects during the run-time and can be used to implement AnyParty pattern.

Stable MVM pattern provides hooks for integration with many other important frameworks, such as:

- **Web Services Architectures and Interactions:** Web services are platform-independent and language-independent and can be defined as a self-describing, self-contained, and a single component of yjr application dynamic that provides specific services through AnyMedia (i.g. Internet). Keywords: Web Services, Wireless Devices, UDDI, WSDL, XML, SOAP, Transport protocols like HTTP, FTP, SMTP, or Message Queues, Flow languages, such as WSFL (IBM) or XLANG (Microsoft), Java Web Service Model.
- **Transcoding:** Transcoding is a realtime content transformation that allows AnyApplication to tailor its models and services for AnyMedia (i.g., web device, hand-hold device, mobile phone) based on their capabilities and resources. Keywords: Device Profile, CC/PP, RDF, XML, XSL,
- **Personalization** allows the application's services to customize their behaviors for AnyParty based on their preferences. This means that personalization is to provide the right AnyModel / AnyView (i.e. information or data) to the right AnyParty (i.e., person) at the right time. Keywords: UIML, XML, AnyParty's preferences, device profile, AnyParty's location.

3. Identifying Device Capabilities.

To map a model, or a collection of models, into a view that suits a specific client, it is necessary that we learn about the capabilities of this client. When this mapping is done in the server or proxy, there must be a way by which they can learn about the capabilities of their client. One way of doing so is through the use of the request header in the HTTP, which is the conventional way of accomplishing this task. However, different schemes were recently proposed for facilitating the identification of the client's capabilities, such as: the Composite Capability / Preferences Profile (CC/PP), the WAP User Agent Profile (UAPROF) standard, and the SyncML Device Information standard (DevInf) [16]. With the different limitations, advantages, and disadvantages of each schema, such implementation decisions should be left to the application designers who can choose the most suitable schema according to their need. Therefore, this implementation issue was not considered in the high level architecture of the pattern shown in Figure 2 before.

Related Patterns

One common approach that has been suggested is to use the *model-view-control* (MVC) framework to address some of issues regarding multi-device interfaces. Developers would prefer a solution that separates the presentation from the application logic. The developer would generate just one application logic or model, and then generate a single view for a single device. This would have to be explicitly addressed in any framework

that embarks on the multi-device challenge. Some of the approaches based on the MVC pattern are given in [1, 2, 3] [San Francisco, and Java platform for wireless Applications: J2EE and J2ME]. Others approaches propose the use of different design patterns such as: Pipe and Filter patterns [13]. However, none of the current solutions consider the situation of having multiple models within the application. Another important issue not being addressed is a way by which an arbitrary numbers of views can be generated from a single abstract model. All existing solutions assume a predefined set of views associated with a single model. In addition, the current solution does not consider the composition of multiple models in a single view and vise versa.

3. CONCLUSIONS

This paper presents the Stable MVM as a collection of patterns that deal with the problem of mapping an abstract model to an abstract view. The Stable MVM pattern handles the situation of having more than one model within the same application. In addition, the generality of the pattern makes it possible to deal with mapping problem for different devices with different capabilities.

REFERENCES

- [1] <http://www-3.ibm.com/software/ad/sanfrancisco/sfclient.html>
- [2] <http://java.sun.com/blueprints/earlyaccess/wireless/designing/designing3.html>
- [3] <http://archive.devx.com/upload/free/features/javapro/2002/04apr02/dh0402/dh0402-2.asp>
- [4] D.C. Schmidt, M.E. Fayad, and R. Johnson, "Software Patterns", *Communications of the ACM*, Vol. 39, No. 10, October 1996, pp. 37-39.
- [5] E. Gamma et al., "*Design Patterns: Elements of Reusable Object-Oriented Software*", Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, New York, 1995.
- [9] M.E Fayad, "Accomplishing Software Stability". *Communications of the ACM*, January 2002/Vol. 45. No. 1
- [10] M.E. Fayad, A. Altman, "Introduction to Software Stability", *Communications of the ACM*, Vol. 44, No. 9, September 2001.
- [11] M.E. Fayad, "How to Deal with Software Stability". *Communications of the ACM*, Vol. 45. No. 4, April 2002
- [12] M.E. Fayad and S. W., "Merging Multiple Conventional Models into One Stable Model", *Communications of the ACM*, Vol. 45, No. 9, September 2002.
- [13] F. Buschmann, et al., *A System of Patterns: Pattern-Oriented Software Architecture*, Wiley and Sons, New York, 1996.
- [14] Froehlich et al. Reusing Hooks, Chapter 9 in *Building Application Frameworks: Object-Oriented Foundations of Frameworks Design*, M.E. Fayad, D. C. Schmidt, and R.E. Johnson (Eds), New York: Wiley & Sons, 1999.
- [15] Dirk Bäumer, Dirk Riehle, Wolf Siberski, and Martina Wulf. *Role Object Patterns*, PLoP 1997.
- [16] Mark Butler, "Current Technologies for Device Independence." HP Laboratories Bristol HPL-2001-83, April 4th, 2001