ABSTRACT

Before heterogeneous devices will be able to access the same application, a mechanism for generating multiple views within a single model is needed. The need for mobile devices to access such applications raises other complications as the resources and capabilities of current mobile devices must be considered in the design of the application itself. 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 visa 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 visa 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. Because models and views will be exchanged over a wireless environment, challenges of synchronization and timing between models and views are introduced.

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. Figure 1 shows the layout of the SSM. 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 [6,7,8,6,12].

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 vise versa. The proposed architecture confronts the limitations of current approaches by providing a flexible mapping between any models to any views and vise 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.

The remainder of the paper is organized as follows: Section 2 describes and documents the Stable MVM pattern, Section 3 describes the collection of the patterns that compose the Stable MVM pattern, and the conclusions are presented in Section 4.

2. STABLE MODEL-VIEW-MAPPING PATTERN DESCRIPTION

Pattern Name: Stable Model-View-Mapping pattern (MVM)

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 vise versa. Most (if not all) solutions to this problem which are based on design patterns, such as observer, proxy, adaptor, mediator [5], master-slave, pipe and filters, forward-receiver, model-view-controller (MVC) [13], and others fall short of delivering a complete and stable solution to the problem of mapping single or models to single or multiple views and vise versa. Because of many possible properties of model-views relationships [17], the issues that govern the mapping mechanisms, such as separation of concerns, abstraction, selection, specialization, integration, and the various level of complexity of mapping from models to views and vise 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.

**Participants**

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.

**CRC- Cards**

<table>
<thead>
<tr>
<th></th>
<th>Responsibility</th>
<th>Collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AnyModel</strong> (Model Descriptor or viewed object)</td>
<td>Describes the abstract model under consideration.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Clients</strong></td>
<td><strong>Server</strong></td>
</tr>
<tr>
<td></td>
<td>Mapping</td>
<td>representModel()</td>
</tr>
<tr>
<td></td>
<td>AnyParty</td>
<td>defineModelProperties()</td>
</tr>
<tr>
<td></td>
<td>Searching</td>
<td>background()</td>
</tr>
<tr>
<td></td>
<td>AnyApplication</td>
<td>modelConstruct()</td>
</tr>
</tbody>
</table>

|                  | **Clients**                 | **Server**            |
| **AnyView** (View Descriptor or view object) | Describes the view under consideration. |                      |
|                  | **Clients**                 | **Server**            |
|                  | Mapping                     | representView()       |
|                  | AnyParty                    | simpleView()          |
|                  | Searching                   | expandedView()        |
|                  | AnyApplication              | defineViewProperties()|
## Stable Model-View-Mapping Architectural Pattern

### DRAFT

#### AnyMedia (Media Descriptor)

<table>
<thead>
<tr>
<th>Responsibility</th>
<th>Collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Represents the media over which the views and models are exchanged and mapped.</td>
<td></td>
</tr>
<tr>
<td><strong>Clients</strong></td>
<td><strong>Server</strong></td>
</tr>
<tr>
<td>Searching</td>
<td>defineMediaProperties()</td>
</tr>
<tr>
<td>Mapping</td>
<td>identifyMedia()</td>
</tr>
<tr>
<td></td>
<td>mediaCapability()</td>
</tr>
</tbody>
</table>

#### AnyParty (Modeler)

<table>
<thead>
<tr>
<th>Responsibility</th>
<th>Collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Models the domain of the problem.</td>
<td></td>
</tr>
<tr>
<td><strong>Clients</strong></td>
<td><strong>Server</strong></td>
</tr>
<tr>
<td>AnyView</td>
<td>model()</td>
</tr>
<tr>
<td>AnyModel</td>
<td>fillModelProperties()</td>
</tr>
<tr>
<td></td>
<td>modelGeneration()</td>
</tr>
<tr>
<td></td>
<td>editModel()</td>
</tr>
<tr>
<td></td>
<td>maintainModel()</td>
</tr>
</tbody>
</table>

#### AnyParty (Viewer)

<table>
<thead>
<tr>
<th>Responsibility</th>
<th>Collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provides the Views of specific model</td>
<td></td>
</tr>
<tr>
<td><strong>Clients</strong></td>
<td><strong>Server</strong></td>
</tr>
<tr>
<td>AnyView</td>
<td>view()</td>
</tr>
<tr>
<td>AnyModel</td>
<td>zoom()</td>
</tr>
<tr>
<td>Searching</td>
<td></td>
</tr>
</tbody>
</table>

#### AnyParty (Requester)

<table>
<thead>
<tr>
<th>Responsibility</th>
<th>Collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests a passive model/view or service.</td>
<td></td>
</tr>
<tr>
<td><strong>Clients</strong></td>
<td><strong>Server</strong></td>
</tr>
<tr>
<td>AnyModel</td>
<td>requestModel()</td>
</tr>
<tr>
<td>Searching</td>
<td>requestService()</td>
</tr>
<tr>
<td></td>
<td>specifyLocation()</td>
</tr>
</tbody>
</table>

#### Mapping (Mapping Handler)

<table>
<thead>
<tr>
<th>Responsibility</th>
<th>Collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maps models to their views and vise versa.</td>
<td></td>
</tr>
<tr>
<td><strong>Clients</strong></td>
<td><strong>Server</strong></td>
</tr>
<tr>
<td>AnyMedia</td>
<td>defineMapRules()</td>
</tr>
<tr>
<td>AnyView</td>
<td>map()</td>
</tr>
<tr>
<td>Applicability</td>
<td>integrate()</td>
</tr>
</tbody>
</table>

#### Searching (Search)

<table>
<thead>
<tr>
<th>Responsibility</th>
<th>Collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Applicability with illustrated example
The applicability of this pattern spans a wide spectrum, ranging from very simple examples, where a simple application contains few passive models that can be mapped in a straightforward way, to a very complex applications that contains several models of different formats (audio, video, text, etc.). In this section the applicability of the proposed pattern is demonstrated through an example.

Example: Weather Application
Problem Description
The objective of this application is to show how the Stable MVM pattern can be used for more complex applications where the user requests an application that consists of many models. In this application, the user wants to view the weather on different devices with different capabilities. A complex weather application has been developed. This weather application might contain, for example, different options and information that the user might be interested in viewing. For instance, the application might contain: the option of getting the weather for all countries and their cities all over the world, warning reports for the next 24 hours, the local weather forecast for the next few days, current satellite, audio and video for the current forecast, etc. Such complex weather applications should be developed in terms of smaller models, since having one model that deals with all of these features will be hard to manage and adapt for different devices. Therefore, it is possible to break the application into several models; Figure 2 shows four possible models that capture some of the weather application features.
The information to be displayed to the user will depend on the device capabilities the user uses to access the weather application. For instance, users who access the weather application from a PC will be given the option to select the country he wishes to view by clicking on a map. On the other hand, for the user who accesses the weather application from a mobile phone, it might be more convenient for him to select the country through a drop down menu.

---

**Figure 2. Weather Application Sample Models**

- **Weather Video**
  - A Signature of El Nino
  - Snow and Ice from the Plains to the Mid-Atlantic

- **Weather Text**
  - Warming in the Southeast, a new cold blast North Plains
  - Sunny skies and highs only slightly below average Cold blast Northern Plains & Lakes
  - A winter storm watch has been issued for the Upper Peninsula of Michigan from east of Marquette to just

---

**Role-based Instance for the Mapping Pattern**

Figure 3 describes the symbols that are used in the Role-based instance diagram. The role-based instance diagram for the searching application is shown in Figure 4.

---

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern Name</td>
<td>Represents the pattern name as given in the pattern object diagram.</td>
</tr>
<tr>
<td></td>
<td>Points to an instance of the pattern in the current application example.</td>
</tr>
<tr>
<td></td>
<td>Represents the typical flow of the interactions within the application example.</td>
</tr>
<tr>
<td></td>
<td>Represents a logical interaction between the different pattern within the application example.</td>
</tr>
</tbody>
</table>
Role-based Instance for the Stable MVM Pattern

The role-based instance diagram for the weather application is shown in Figure 4.

Role-based Scenario for the Stable MVM Pattern for the Weather Application

Based on Figure 4, the searching application can be described in terms of each component in the Stable MVM pattern. [The pattern names are written in bold italic. The roles are written in bold and underlined]

The developer (AnyParty) generates different models (AnyModel) that construct the weather application. Such models could be: a map with different colors, different video clips, plain text, etc.

Suppose a user (AnyParty) requests the weather application using his PC (AnyMedia) through the Internet (AnyMedia). The server receives the user request and defines the capabilities of the user’s client (the PC in this example). Based on the client capabilities and, perhaps, the user preferences, the features of the required views are determined (Applicability). For instance, in this example the required view will be a combination of different models; therefore, integration is one possible applicability in this case.

The mapping properties (Mapping) get the suitable models (AnyModel) and map it into the relevant view (AnyView). The generated view will be sent back to the User (AnyParty) through the Internet (AnyMedia). The User (AnyParty) views the received view on his PC (AnyMedia).
Stable Model-View-Mapping Architectural Pattern

AnyModel → Mapping

ApplicationType

Integration → Simplification

Mapping → User

Model 1 → Mapping

Server

Forward request

Return view

Generate

View

Request

Wireless Network

AnyView

User

AnyMedia

User

AnyApplication

Search

AnyParty

Generate

Developer

AnyModel

Weather Video

Photo Model

Map Model

Weather Text

Weather Video Model

Text Model

Simplification

Generate

Request

Search

Search

Send

View
Figure 4. The Instantiation of the Stable MVM Pattern into the Weather Application

Figure 5 below shows how the integration of the different models of the weather application can form a possible view.

Figure 5. A Possible View for the Weather Application
[Source: Yahoo Weather Web Page]

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 Model-View-Mapping Architectural Pattern**

**DRAFT**

**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 your application dynamic that provides specific services through AnyMedia (i.e., 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.e., 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.

**Consequences**

- The use of the Stable MVM pattern has the following benefits:
  1. **Handle more than one model.** Unlike the current solutions for model-view problem, the Stable MVM pattern does consider the situation of having more than one model within the same application. This is done through the use of the AnyModel class.

  2. **Handle Different Device Capabilities.** The Stable MVM is general enough to handle the diverse capabilities of different devices. With such diversity in device capability, the mapping properties should be adjusted in order to best map the model(s) into the appropriate view.
Consider Different Media Nature. The Stable MVM pattern considers the mapping between models and views over different media natures. This is achieved by the AnyMedia pattern, which specifies the media type and its possible kinds (See Appendix for pattern description). This feature increases the flexibility of the pattern since the mapping problem is needed in vast applications with different natures.

Adaptable for Required Goals. The Stable MVM pattern maintains a high level of adaptability to best achieve the goals of performing the mapping between models and views. The mapping process depends on the goals of conducting this mapping. For instance, mapping between models and views with simplicity as a goal will enforce specific properties that the mapping process should follow; therefore, the generated views will match the goal in mind. The determination of the application type will adapt the Mapping pattern to best meet the goals of this mapping.

- The use of the Stable MVM pattern has the following limitations:
  1- Hidden Patterns Description. At first glance, it would be hard to capture all of the underlying issues that are related to the different patterns that composite the main Stable MVM pattern. For instance, issues related to the mobility issues are not seen in the main MVM pattern; however, these issues should be considered within the AnyMedia pattern details.
  2- No Industrial Objects to Clarify Pattern Applicability. Since the Stable MVM has been developed based on software stability concepts, there are no IOs attached to the pattern itself, which makes the pattern’s applicability not very obvious from just reading the MVM pattern structure. However, attaching such IOs (which are implementation details) will narrow the applicability of the pattern. Showing detailed case studies for the pattern applicability make the pattern usage obvious; yet, preserve the generality of the main pattern.

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 vice 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 provides several advantages over the existing solutions. It 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. The pattern also considers the mapping over the different media natures, which increases the flexibility of the pattern since the mapping problem is needed in vast applications with different natures.

REFERENCES
