Software System Engineering

Dr. M.E. Fayad, Professor
Computer Engineering Department, Room #283I
College of Engineering
San José State University
One Washington Square
San José, CA 95192-0180
http://www.engr.sjsu.edu/~fayad

© M.E. Fayad 1996-2006
SJSU -- CmpE
Lesson 7-1: Stable Analysis Patterns
Lesson Objectives

- Explore Stable Analysis Patterns
- Examine Stable Analysis Patterns
- Discuss the following Stable Analysis Patterns:
  - Negotiation
  - Model-View-Mapping (MVM)
  - Searching
The concept of extracting patterns using software stability concepts

- Problem

**FIRST ABSTRACTION:**
- Apply SSM concepts
- Define EBTs, BOs, and IOs
- Create SSM for the problem

**SECOND ABSTRACTION:**
- Model each EBT using SSM concepts
- Build natural domain Stable Analysis Pattern
Stable Analysis Patterns’ Properties

- Timeless Notion Pattern
- Generality: Generality Patterns
- Usability and Reusability Patterns
- Scalability Patterns
- Presentation of the Core Knowledge Patterns
- One Presentation of the Problem Space Patterns
- Configurability and Re-Configurability Patterns
- Adaptability Patterns
- Extensibility Patterns
- Accessibility Patterns
- Integratability Patterns
Stable Analysis Patterns Advantages

- Contribute to Stable Architectures
- Aid True Problem Understanding
- Provide True Problem Modeling
- Enhance Team Dynamic
Negotiation Analysis Pattern (1)

- Negotiation is a general concept that has many applications in various contexts.

- In our every day life, there are various situations where negotiation usually takes place. For instance, buying or selling properties usually involves some sort of negotiation (e.g. buying or selling a home or a car).

- In addition, negotiation can be used for solving social and political conflicts.
Negotiation Analysis Pattern (2)

- In software systems, Negotiation also appears frequently in the development of different applications.

- For instance, developing software for online auctions and shopping might involve the negotiation of the price and/or the negotiation of different product aspects.

Negotiation Analysis Pattern (3)

- Today, devices that need to access the Web diverge greatly in their capabilities, making it highly desirable for the same resource to be available in several different representations (different languages for example).

- Negotiation algorithms play a fundamental role in aiding servers to decide which representation of a document a device should be given. In this case, the browser (or the client agent) will indicate its preferences by including a header in the request.
Negotiation Analysis Pattern (4)

- Pattern Name: Negotiation

- Problem: How does one build a negotiation model that can be used to model the negotiation problem in any application?
Forces (1)

- Negotiation spans many contexts that are completely different in their natures.

- The Negotiation process can take place between two or more persons, between persons and organizations, or between two non-human entities; in each case, the negotiator structure is completely different. How can we handle these different structures using a single model?
The Negotiation entity can be an organization consisting of many persons, each having a role in the negotiation process. For instance, there can be one individual who is responsible for negotiating financial issues, another who is responsible for negotiating issues related to management, and so on. Therefore, our pattern should be flexible enough to handle different negotiator structures.
Forces (3)

- Negotiation can be conducted through one or more media either simultaneously or consecutively; thus, the pattern should handle the use of multiple media.

- Negotiation can be performed on one or more affairs at the same time. For instance, negotiation that takes place in the buying and selling context usually involves more than one subject to be negotiated. For example, in buying a car, one can negotiate the price, the warranty, and so on. Therefore, the pattern should be flexible enough to handle such situations.
The ultimate goal of any negotiation is to reach an agreement between the negotiators. However, the nature of this agreement varies tremendously from one application to another and from one context to another. An agreement that might be reached while negotiating a political conflict is completely different from that reached while buying a car. Therefore, the pattern should be able to handle these wide variations.
Pattern Structure and Participants
## CRC Cards (1)

### AnyNegotiation (Negotiation Descriptor)

<table>
<thead>
<tr>
<th>Responsibility</th>
<th>Collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Describes the negotiation</td>
<td></td>
</tr>
<tr>
<td>rules and regulations to the</td>
<td>AnyAgreement</td>
</tr>
<tr>
<td>negotiating parties.</td>
<td>AnyParty</td>
</tr>
<tr>
<td></td>
<td>AnyMediaAnyContext</td>
</tr>
<tr>
<td></td>
<td>defineRules()</td>
</tr>
</tbody>
</table>

### AnyParty (Negotiation handler)

<table>
<thead>
<tr>
<th>Responsibility</th>
<th>Collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performs and finalizes the</td>
<td></td>
</tr>
<tr>
<td>negotiation.</td>
<td>AnyNegotiation</td>
</tr>
<tr>
<td></td>
<td>AnyAgreement</td>
</tr>
<tr>
<td></td>
<td>negotiate()</td>
</tr>
<tr>
<td></td>
<td>approve()</td>
</tr>
<tr>
<td></td>
<td>agree()</td>
</tr>
</tbody>
</table>
## CRC Cards (2)

### AnyAgreement (Agreement)

<table>
<thead>
<tr>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Describes agreement terms and conditions.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clients</td>
</tr>
<tr>
<td>Server</td>
</tr>
</tbody>
</table>

- **AnyNegotiation**
- **AnyParty**
  - option()
  - sign()
  - execute()
  - insert-option()

### Media (Connector)

<table>
<thead>
<tr>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communicates negotiation issues between negotiators.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clients</td>
</tr>
<tr>
<td>Server</td>
</tr>
</tbody>
</table>

- **AnyNegotiation**
  - connect Parties()
  - display()
  - illustrate()
### Context (Motivator)

<table>
<thead>
<tr>
<th>Responsibility</th>
<th>Collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defines the reason of the negotiation process.</td>
<td>Clients</td>
</tr>
<tr>
<td><em>AnyNegotiation</em></td>
<td>defineBoundary() contextLimit()</td>
</tr>
</tbody>
</table>
STABLE MODEL-VIEW MAPPING PATTERN (MVM)
Outline

- Problem Overview
- Solution Requirements
- Current Solutions – Overview
- Current Solutions - Limitations
- Proposed Solution
- Stable Model-View Mapping [MVM] Pattern
- Stable MVM- Second Abstraction Level
- Stable MVM Pattern Instantiation
- Applicability Examples
Problem Overview (1)

- Today, very heterogeneous devices are required to access the same applications.
- PCs, PDAs, WAP phones, and interactive TVs, each has its own limitations and capabilities.
- Generating different version of the same application is impractical solution. [Cost, management, time, etc.]
**Problem Overview (2)**

**Problem:** How to map models to different views!

- **View 1:** The Internet
- **View 2:** Wireless Network
- **View 3:** Server
- **View 4:** Application 1
- **View 5:** View 6

---

© M.E. Fayad 1996-2006

SJSU – CmpE

M.E. Fayad

L7-1-S22 Stable Analysis Patterns
Solution Requirements (1)

- Successful applications will be accessible from several devices.
- Complex applications should be built as a composite of multiple models, not as a single complex model.
- Therefore, the solution should deal with the situation of having multiple models within the applications.
Solution Requirements (2)

- Sometimes the required view is constructed from the isolation of different parts from several models within the application. Mapping mechanisms are not predefined.

- The pattern needs to be abstract and not tied to any specific technology.
Current Solutions - Overview

- Developers would prefer a solution that separates the presentation from the application logic.
- 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. [For example: 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.
Current Solutions- Limitations

- None of the current solutions consider the situation of having multiple models within the application.
- Therefore, they do not consider the composition of multiple models in a single view and vise versa.
- None of them have addressed the 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.
Proposed Solution (1)

- We propose the Stable Model-View Mapping (MVM) as a solution to the mapping problem.
- The Stable MVM is a collection of patterns that deal with the problem of mapping an abstract model to an abstract view.
- The pattern handles the situation of having more than one model within the same application.
- The pattern is general enough to deal with mapping problem for different devices with different capabilities.
Proposed Solution (2)

- No predefined models and predefined views
- Mapping of Passive Models, such as relations in RDB and Live Models, such as return of `playMovie()`.
- The pattern is applicable for multiple purposes, such scalability, simplicity, viewing, fitting, editing, etc.
- The pattern provides a base for building multiple applications.
Participants (1)

- The participants of the Stable MVM pattern are divided into two kinds: **Classes** and **Patterns**.

**Classes**
- **Applicability.** Describes the application and the purpose for which mapping is needed.

**Patterns**
- **AnyModel.** Describes the models within the application. The model is a representation of the data within the application.
Participants (2)

- **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.** 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 (Phone, PCs, etc.)
Participants (3)

- **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.
2nd Level Of Abstraction

How to build a model that can be used to present any media in any application?
Applicability with Illustrated Example

Weather Application

- The weather application provides several services, such as: the weather for all countries and their cities all over the world, the weather forecast for the next few days, satellite photos, current forecast on video, etc.
- In this application, the user wants to view the weather on different devices with different capabilities.
- Weather applications is 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.
Weather Application (1)

- Interactive map model
- Photo model
- Video model
- Text model

One possible view
For the weather application

Weather application view on the PC

- Weather information
- Warning reports
Weather Application (2)

Weather Application

Interactive map model

Photo model

- Weather information
- Warning reports

Video model

Text model

One possible view
For the weather application
Weather Application (3)

- To show the applicability of the Stable MVM pattern to the weather application, we show an instance of the pattern.
- The pattern instantiation is useful in visualizing each component in the pattern in reality.
- The instantiation diagram shows also how the different components of the pattern interact with each other in real applications.
### Instantiation Symbols Description

<table>
<thead>
<tr>
<th>AnyView</th>
<th>Represents the pattern name as given in the pattern object diagram</th>
</tr>
</thead>
<tbody>
<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>
Weather Application

• Weather information
• Warning reports

AnyModel

Video model

Text model

AnyMedia

Mapping

Applicability

Integration

Simplification

Searching

View

Use

Forward request

Forward view

Forward request

Forward view

Forward request

Forward view

Generating

SJSU – CmpE

M.E. Fayad

L7-1-S39

Stable Analysis Patterns
Discussion Questions

1. Generate instance diagrams for another applications of MVM.

2. Model the following concepts: viewing, simplifying, searching, and scalability.

3. Use negotiation pattern to model the following concepts: diplomatic negotiation, negotiating peace settlement, and trade negotiation.