Automation and Testing of Software Design Pattern for e-commerce Web Application Development using J2EE MVC Architecture

Vedavyas J, Kumarswamy Y

Abstract- The Model-View-Controller design pattern is cited as the architectural basis for many J2EE web development frameworks. Here analysis of those changes, and proposes a separate Web-MVC pattern that more accurately describes how MVC is implemented in web frameworks. The MVC is very useful for constructing dynamic software systems. Partitioning decisions can be changed without modifying the application. The most important part of the design pattern is to build the reusable and well structured software. Thus it became worthy to detect which design patterns are present in the software system. Approaching this necessitate, techniques for automated design pattern detection have appeared in this paper. These applications using patterns help to reduce the maintenance costs and ease the creation of the new tests. Each pattern consists of definition and details highlighting its suitability for e-commerce test automation.

Index Terms: Automation, Controller, Design Patterns-commerce, Framework, Model, Partitioning, View

1 INTRODUCTION

MVC is the design pattern for the architecture of web applications. Many languages have implemented the frameworks and adopted them universally. Basics Components of MVC model.

1. Model: business logic & processing
2. View: user interface
3. Controller: navigation & input

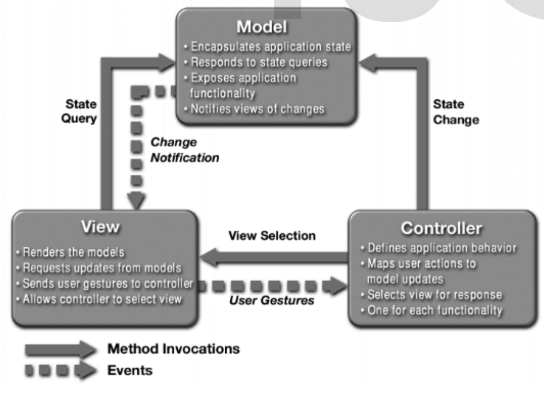

Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
State Change
Change Notification
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events

Fig 1.Generic MVC Structure

MVC Design patter is one of the most fundamental architecture for web applications like J2EE, .Net, Rails and Struts etc.
J2EE and JSP technologies are the fundamentals for struts 2 framework. This Struct2 framework consists of MVC pattern as follows.

a. User Interface component as views
b. Application logic as Model
c. Control functions as Controller

The view component of strut 2 framework is done by embedding JSP tags which provides diversified functionalities like flow control, accessing model component and effectual HTML forms structures. The controller component is corporeal with java classes for the actions to be implied. Each action has a responsibility to validate user Input and to engineer transaction processing by invoking appropriate model operations.

The XML configuration file or the java annotation mechanisms are used to outline and configure the actions. Such information is used to control the flow of web applications by finding the outcomes of each action. Value stack eliminates much of the tasks involved in handling HTTP requests and provides the information for JSP to display. It is the key factor which contains the information between view and controller and converts when needed.

This section describes how the MVC is being represented in the web application frameworks. It also reflects the evolutionary changes in the web frameworks.

The primary responsibilities of MVC-Web model are:

1. It has to maintain a database for the data persistence
2. It has to execute the application logic that operates on the application state called transaction processing.
3. It has to manage interactions with external agents such as web services known as External Interface.
4. It should handle query to provide the information to view and controller elements in response for queries.

The primary responsibilities of the view component are:

1. It is used for information retrieval and display because it displays information to the user based on the query in the model.
2. It provides input forms and controls for the user to interact with the application.
3. It provides interactive dynamic behavior at the client side for the users.

The primary responsibilities for the MVC-Web Controller are:

It receives the incoming request and routes them to appropriate handler.

It receives the request parameters and handles the action such as invoking appropriate model elements.

It provides the response for the request depending upon the action invoked.

2 J2EE WEB APPLICATION: AN EXAMPLE ON PARTITIONING

Let us consider the web-application where a client wants o fetch information about a company's employees in a simple way by executing two operations.

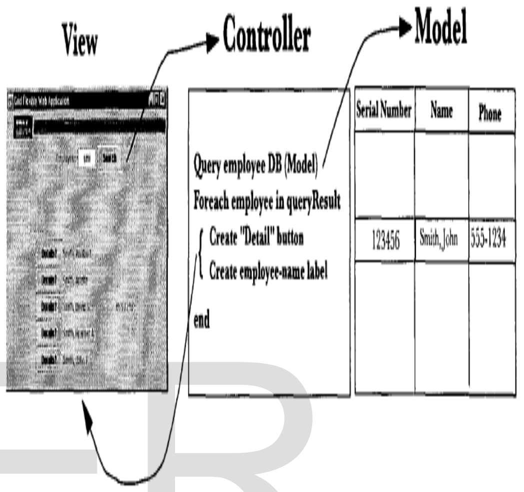
1. By supplying a name, and clicking on a "search" button, search the employee directory "by name". The search returns the set of employees that match the search criteria in a format that displays an abbreviated employee record for each member of the returned set.

2. By clicking on a "details?" button, get detailed information about a specific employee. Implementation in a stand-alone, single address-space, environment, is straightforward. From the perspective of the MVC design pattern (see Figure 1):

The Model consists of the records in the employee directory. There are four Views: a "search" panel; a display of abbreviated information about a set of employee records; a display of detailed information about a specific employee; and a report that no employees match the search criteria.

There are two Controllers: one that, given a "search" directive, drives the process of querying the Model and returns a result set; and one that, given a "details" directive, queries the Model to get the full set of information about the specified employee. Implementation as a web-application in a server environment raises the issue of partitioning which is conceptually irrelevant to, but in practice complicates, the MVC design pattern. Naively, as there are two Controllers, the application can be implemented in one of four ways.

Either both Controllers execute exclusively on the client or server, or one Controller executes on the client and the other executes on the server. Each partitioning decision greatly affects the way that the application is implemented. For example, if both Controllers run on the client (the "fat-client" approach), the entire Model must be downloaded to the client -- which is often impractical. If both Controllers run on the server (the "thin-client" approach), two round trips between client and server must be performed each time that the client searches for an employee and then asks for more detail about that employee.



```
Query employee DB (Model)
Foreach employee in queryResult
  { Create "Detail" button
  { Create employee-name label
end
```

Fig 2. MVC for Employee Record.

In fact, for many environments, either the thin-client or the fat-client is ideal. Instead, using a dual-MVC approach, we partition the Controllers between the client and server. Specifically, the "search" Controller executes on the server in association with a Model consisting of the complete employee directory. However, when returning relatively small sets of employee records, the Controller also returns the full record for each of the employees, so that they can be maintained in the client-side Model. The dual-mvc approach allows requests for detailed employee information to be served by the client, thus eliminating a client/server interaction. (This implementation is beneficial only when application scenarios typically consist of a preliminary search for an employee using a "partial name", followed by request for more information after the specific employee is determined by inspection. Remember: this is only a motivating example!)

Of course, what we really want is to do avoid partitioning while implementing the application, since the correct partitioning decision depends on factors that are not necessarily determined until actual deployment. For example, if the employee directory is relatively small, the "fatclient" approach with both Controllers executing on the client makes sense and would provide better performance.

Conversely, if the application is deployed in a "internet" environment in which users want minimal customization of their environment, the "thin-client" approach may be the only solution possible. Delaying application partitioning for as long as possible is even more attractive because partitioning gets in the way of designing the Views and developing the business logic needed by the Controllers. Flexible web-application partitioning addresses these needs. In fact, flexible web-application partitioning goes further, allowing partitioning decisions to vary dynamically, during application execution.

The J2EE programming model explicitly supports the MVC design pattern, and enables programs executing in MVC mode to execute in a single address space. When deployed, these programs can be flexibly partitioned without changing the source code used during smvc development. We refer to such J2EE applications as J2EElications.

3 INTRODUCTION TO AUTOMATION AND TESTING OF DESIGN PATTERNS

In this today's world delivering knowledge and experience from veteran to recruit is by design patterns which the fundamentals are of object oriented design heuristic. These design patterns are well designed class libraries used for abstracting and delivering its basics. The problems faced in the manual application of prototype and observer design patterns are often time consuming and technical, hence an error prone activity.

Design pattern in automation has no proper formation within the reach so that actor participate in pattern implementation faces problems.

The followers of the Alexandra feel that use of natural language prose as the principal means of pattern specification. As per the standards of the software engineering the natural languages bear to have vagueness and ambiguity.

Algorithms are used to specify the design patterns which are the only natural to use a programming language by which design patterns are expressed. By using the metal language the design patterns are treated as programs that manipulate other programs.

Automated testing provides various gains to the organization like finding defects and cost effective in the development process.

The design and automation of tests is an art as well as science which should be take care by the writer. Writer should keep the common principles of software development like simple design, tuning of design patterns which are underuse in the test development.

In this paper we are implementing the unit testing by using a very simple call to methods for checking the expected results because of the complication in the tests. We have chosen the single test to prove its results instead of screening multiple screens.

4 DESIGN PATTERNS

"A pattern is a named problem /solution that can be applied in new contexts." One of the most important features of a pattern is that it can be reused for many times in various fields.

4.1 Classification of Design Patterns

1) Creational Design Pattern: Abstract the instantiation process. They help make a system independent of how its objects are created, composed, and represented. Depends more on composition than inheritance. The emphasis shifts away from hard coding a fixed set of behaviors toward defining a smaller set of fundamental behaviors that can be composed into any number of complex ones. Abstract Factory, Builder Pattern

2) Structural Design Pattern: Concerned with how classes and objects are composed from larger structures. Useful when independently developed class libraries work together. Adapter, Composite

3) Behavioral Design Pattern: Behavior patterns are concerned with the algorithms and the assignment of responsibilities between objects. These patterns characterize the complex control flow that's difficult to follow at run-time. You concentrate more on the way objects are communicated. Template and Interpreter pattern Test Automation is software used to control the execution of tests, comparing actual vs. predicted results, setting up the test preconditions and other controls.

4.2 Levels of Automation

1. Unit Automation
2. Integration Automation
3. User Interface Automation
4. Web Service Level

In this paper we are going to solve some of the following issues such as

1. Maintainability
2. Reusability
3. Availability of time
4. Reliability
5. Modularization

5 CASESTUDY

Scenario for searching a product and verifying that price of product starts with $

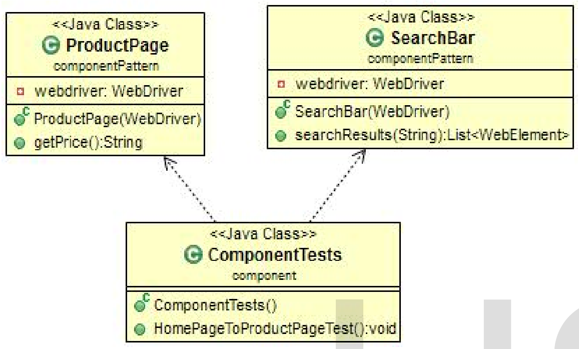


Fig. 3. The UML Class diagram showing ProductPage an d searchBar with ComponentTests.

SearchBar Module

```
public List<WebElement> searchResults(String word)
{
webdriver.findElement(By.id("search")).sendKeys(word);
Webdriver.findElement(By.id("srchBtn")).click();
List<WebElement> products =
this.webdriver.findElements(By.id("productId"));
return products;
}
```

ProductPage Module

```
public String getPrice()
{
String productPrice =
this.webdriver.findElement(By.id("price")).getText();
return productPrice;
}
```

Test Module

```
public void HomePageToProductPageTest()
{
SearchBar searchBox = new SearchBar(driver);
List<WebElement> webelements =
searchBox.searchResults("books");
webelements.get(0).click();
ProductPage productPage = new ProductPage(driver);
Assert.assertTrue(productPage.getPrice().startsWith("$"));
}
```

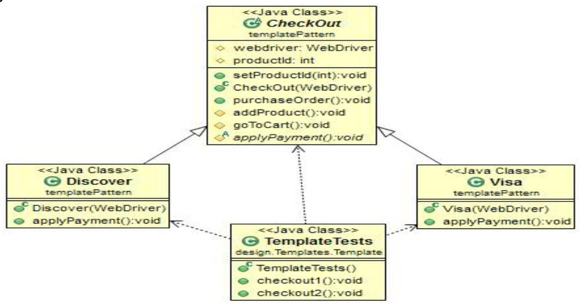


Fig. 4. The UML Class diagram showing CheckOut class.

Checkout Module

```
public void purchaseOrder()
{
addProduct();
goToCart();
applyPayment();
}
protected void addProduct()
{
Webdriver.get("http://www.myshopping.com/"+productId+
"/product.html");
}
protected void goToCart()
{
Webdriver.get("http://wwww.myshopping.com/cart.html");
}
abstract protected void applyPayment();
```

ApplyPayment Module

```
@Override
public void applyPayment()
{
Webdriver.findElement(By.id("visa")).click();
webdriver.findElement(By.id("cardno")).sendKeys("111122223
3334444");
webdriver.findElement(By.id("expmon")).sendKeys("10");
webdriver.findElement(By.id("expyr")).sendKeys("2014");
webdriver.findElement(By.id("submit")).click();
}

@Override
public void applyPayment()
{
webdriver.findElement(By.id("discover")).click();
webdriver.findElement(By.id("cardno")).sendKeys("444433332
2221111");
webdriver.findElement(By.id("expmon")).sendKeys("10");
webdriver.findElement(By.id("expyr")).sendKeys("2014");
webdriver.findElement(By.id("submit")).click();
}
```

Test Module

```
@Test
public void checkout1()
{
WebDriver driver = new FirefoxDriver();
driver.get("http://www.myshopping.com");
CheckOut checkout = new Visa(driver);
checkout.setProductId(123);
checkout.purchaseOrder();
}
@Test
public void checkout2(){
WebDriver driver = new FirefoxDriver();
driver.get("http://www.myshopping.com");
CheckOut checkout = new Discover(driver);
checkout.setProductId(123);
checkout.purchaseOrder();
}
```

The above said common issues are been addressed with solutions

1. Maintainability – Functionality is defined in each component
2. Reusability – Tests call the component
3. Time – Common functionality defined in the components.
4. Reliability – All Tests calling the same component will fail.
5. Modularization – Functionality of each component is defined.

1. 6 Conclusion and Future Work

This paper describes how the partition-independent Model View Controller design pattern can be used in the intrinsically locution-dependent environment of partitioned Web-applications. By understanding the scenario flows, the application can be partitioned in a way that improves performance. In contrast, traditional implementation techniques require that such analysis be performed only in the design and requirements phase because it is much too costly to repartition the application once it is deployed. Unfortunately, the necessary insights can often be made only after the application has been deployed and in production for some time. In future repartitioning, under fwap, imposes no extra cost; an application can therefore be readily tuned after deployment based on feedback from actual client use. We are currently implementing the algorithms and infrastructure needed to enable fwaplications to scale over non-trivial application Models. We are also working with a customer to validate the fwap concepts and implementation. In this paper we presented the narration approach to design pattern automation. The main characteristic of this paper is to simplify and to make intuitively appealing.

Applications are invariable acquiring poses severe sustenance problems for the automated testers. Maintenance problems are dealt by using design patterns and thus dealing test code with the same grandness as application code. This paper presents a class of design patterns that evidence how to alleviate test code reuse, adaptability, changes and maintenance without duplication for surviving tests.

REFERENCES

[1] Shuster, J., UIML: AnAppliance-Independent XML User Interface Language, Proceedings of the Eight International World Wide Web Conference, May, 1999,617-630.

[2] Barracuda: Open Source Presentation Framework, http:// barracuda.enhydra.org/, 2001.

[3] Beck, K., Extreme Programming Explained: Embrace Change (XP Series), Addison-Wesley Pub Co., 1999.

[4] Bennett, B. et al, A distributed object oriented framework to offer transactional support for long running business processes, IFIPIACM Intemational Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware 2000).

[5] Bergsten, Hans, JavaServer Pages, O'Reilly, 2000.
Betz, K., Leff, A., Rayfield, J., Developing Highly-Responsive

[6] User Inte$aces with DHTML and Servlets", Proceedings of the 19th IEEE International Performance, Computing, and Communications Conference – IPCCC-2000, 2000.

[7] Buschmann, F. et al, Pattern-Oriented Software Architecture: A System of Patterns, John Wiley and Sons, 1996, 123- 168.

[8] Coutaz, J., PAC, An Object-Oriented Model for Dialog Design, Elsevier Science Publishers, Proceedings of Human-Computer Interaction - INTERACT, 1987,43 1-436.

[9] Enterprise JavaBeans Specipcations, http://java.sun.com/ products/ejb/docs.html, 2001.

[10] JAVA PLUG-IN 1.2 SOFTWARE FAQ, http://java.sun.com/ products/plugin/l.2/plugin.faq.html , 2001.

[11] Flanagan, David, JavaScript: The Definitive Guide, 3rd, O'Reilly, 1998.

[12] Gray, G and Reuter, A. Transaction Processing: Concepts and Techniques, Morgan Kaufmann, 1993.

[13] JAVA SERVLET TECHNOLOGY IMPLEMENTATIONS & SPECIFICATIONS, http://java.sun.com/products/servlet/download. html#specs , 2001.

[14] Java 2 Platform, Micro Edition (JZME), http:// java.sun.com/j2me/, 2001.

[15] G.E. Krasner and S.T. Pope, A Cookbook for Using the Model-View-Controller User-Interface Paradigm in Smalltalk-80, SICS Publication, 26-49, Journal of Object-Oriented Programming, August/September, 1988.

[16] Struts, http://jakarta.apache.org/struts/index.html2, 001.

[17] WebW ork, http://sourceforge.net/projects/webwor2k,0 01.

[18] Alexander, Christopher, Sara Ishikawa, Murray, Silverstein Max Jacobson,

[19] Ingrid Fixdahl-King, and Shlomo Angel (1977). "A Pattern Language". Oxford University Press, New York.

[20] Eden, Amnon H, Joseph (Yossi) Gil, and Amiram Yehudai (1996) . "A Formal Language forDesign Patterns (extended abstract). A technical report, The Department of Computer Science, School of Mathematics, Tel

Aviv University.
[21] Gamma Erich, Richard Helm, Ralph Johnson, and John Vlissides (1995). Design Patterns:Elements of Reusable Object Oriented Software. Addison-Wesley.
[22] Pal, Partha P. and Naftali Minski (1996). Imposing the Law of Demeter and Its variations.TOOLS USA 1996.
[23] Crispin, Lisa, Tip House Testing Extreme Programming. Addison-Wesley, 2002

- *Vedavyas J, Asst. Prof. Department of MCA, BITM, Bellary, VTU*
- *Kumarswamy Y, Prof. and HOD. Department of MCA, Dayanandasagar College of Engineering, Bangalore, VTU vedavyasjamakhandi@gmail.com yskldswamy@yahoo.co.in*