AUTOMATIC METHOD FOR TESTING STRUTS-BASED APPLICATION

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Shweta Tiwari

In Partial Fulfillment
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

March 2013

Fargo, North Dakota

North Dakota State University
Graduate School

Title

Automatic Method For Testing Strut Based Application

By

Shweta Tiwari

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Kendall Nygard
Chair

Kenneth Magel

Fred Riggins

Approved:

4/4/2013	Brian Slator
Date	Department Chair

ABSTRACT

Model based testing is a very popular and widely used in industry and academia. There are many tools developed to support model based development and testing, however, the benefits of model based testing requires tools that can automate the testing process. The paper propose an automatic method for model-based testing to test the web application created using Strut based frameworks and an effort to further reduce the level of human intervention require to create a state based model and test the application taking into account that all the test coverage criteria are met. A methodology is implemented to test applications developed with strut based framework by creating a real-time online shopping web application and using the test coverage criteria along with automated testing tool. This implementation will demonstrate feasibility of the proposed method.

ACKNOWLEDGEMENTS

I would like to sincerely thank Dr. Kendall Nygard, Dr. Tariq M. King for the support and direction. I would like to thank Stephanie Sculthorp (Administrative Secretary) of the Computer Science department for helping me to complete my paper on time. I would also like to thank my husband Abhishek Misra and my parents who gave me encouragement to complete the paper.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

MVC…………………	Model View Controller
SBFs...........................	Strut Based Frameworks
MBTs..........................	Model Based Testing
SCXML.....................	State Chart Extensible Mark Up Language
JSPs............................	Java Server Pages
UML...........................	Unified Modeling Language

1. INTRODUCTION

Web application has become the most important and the first choice for business application development which is leading to advancement in the automation of web application development and testing. Andrews et al. [11] lists several of the pieces that are connected together during the development of a Web application including: static links, dynamic links, dynamically created HTML pages, user/time specific GUIs, operational transitions, software connections and dynamic connections. To validate, test and support model based testing, the web applications graphical user testing and testing using Finite State Machine(FSMs) has been merged[10]. Frameworks such as Struts[3] are widely used and available and are continually used for faster and easier development. The application architecture created using Struts-Based-Framework uses Model-View-Controller(MVC) designs.

In this paper, Net UI Page flow(or Java Page Flows) is used for implementation. Network User Interface Page Flows or Java Page Flows is a technology that provides a flow-control programming model and framework, called Page Flow, which is based on Apache Struts [1][3]. The Net UI Page Flow is an annotated driven web application framework that uses struts based framework [1][4]. Strut based frameworks aids in design of web application by using graphical representation for the application flow and the graphical representation is called as Page Flow View in Oracle workshop for web logic [7]. This Page Flow view helps us to choose test information useful for the methodology.

The objective of this paper is to implement a methodology to support model based testing by automating the testing process for strut based applications which helps to reduce the level of human intervention. To implement the methodology, a real time online shopping application is developed using Java Page Flow and JSPs, and tested taking into account that all the test coverage criteria has been covered in conjunction with the automated testing tool.

This paper implements an automatic method for validating java page flows by creating state charts to support model based testing. Alva, King and Clark[1] transformed the page flow model into typed attributed directed graphs and the graphs helped to identify the important elements which can be useful to analyze the test coverage criteria defined which included All Pages, All Actions, All Links and All Forwards. I have used the same test criteria- All actions, All pages, All Links and All Events described by Alva, King and Clark[1] approach to test my page flow application.

In Alva, King and Clark[1] and Andrews et al[11] approach they have constructed hierarchical finite state machine(FSMs) to support model based testing. The authors used hierarchical decomposition of the Web applications into clusters, each composed of Web pages and other clusters to construct the FSMs. This hierarchical FSMs model subsystem of the web applications are used to generate random test cases along with the textual input from predefined data pool [1]. In Ricca and Tonella [13][2] approach, they have propose an approach to analyze and test a Web application based on a high level UML model of the Web application. The UML model for each Web application being analyzed is an instance of a meta model the authors developed for a generic web application. The testing approach used by Ricca and Tonella [13][2] is also based on the traditional structural testing criteria.

For my approach to support model based testing, I have created a state chart for the application with the help of SCXML [12]. The SCXML document will describe the behavior of my online shopping page flow application in terms of states, transition and events. This state chart will also cover the test criteria defined. The elements associated from this state chart representation will be used as an input to the automated testing tool and test the navigational flow of the application and the test criteria defined.

SCXML is an XML based markup language which provides a generic state machine based execution environment and is used to define states and behavior of the objects[12]. The state chart defines all actions, all events and all paths for the online shopping web application. The creation of the state chart for the entire web application facilitates the generation of test scripts and supports automation. Generating test scripts using a state chart involves traversing through various combinations of conditions described in the state chart document.

For testing, an automated testing tool is used in conjunction with the state chart to generate test scripts and apply them to system under test. I am using Selenium 2.0 [6] automation testing tool for automated testing and have written a program which will invoke the application, perform actions and relay the user initiated events to the Commons SCXML[5] driven online shopping instance which serves as an intermediary between the UI and application behavior. In state chart the paths are mapped into events that translate into navigation interaction. The successful execution of all pages, all events and all paths results in coverage criteria satisfied

1.1. Overview of Testing Approach

Struts based framework(SBFs) have an underlying navigational model and it aids in the design of web application by using graphical representation for this navigation model .The testing approach I have used in this paper will use the elements from this navigational model. This graphical representation will automatically generated during application development.

Figure 1.1 shows the diagrammatic representation of the testing method I have used to test the SBF application.

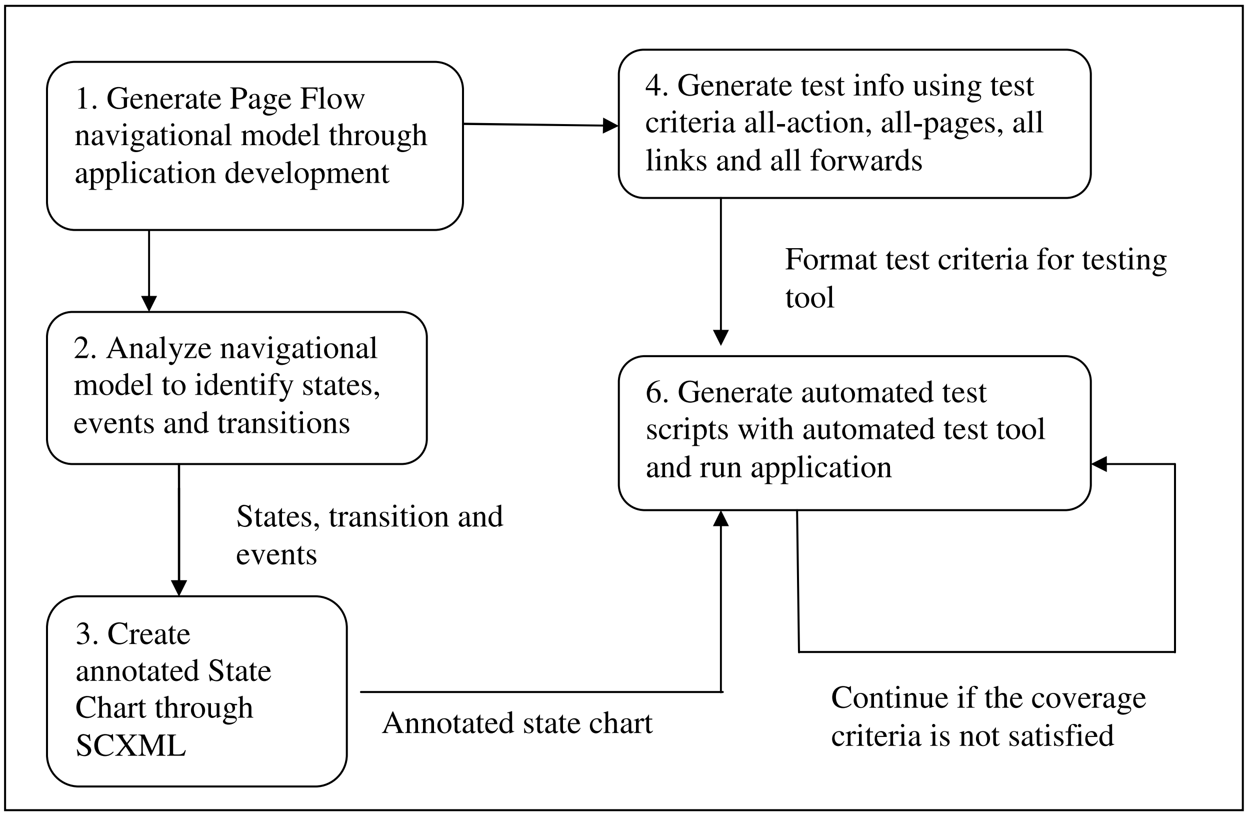

1. Generate Page Flow navigational model through application development
4. Generate test info using test criteria all-action, all-pages, all links and all forwards
Format test criteria for testing tool
2. Analyze navigational model to identify states, events and transitions
6. Generate automated test scripts with automated test tool and run application
States, transition and events
3. Create annotated State Chart through SCXML
Annotated state chart
Continue if the coverage criteria is not satisfied

Figure 1.1. Overview of the Testing Approach Based on Page Flows

The steps of the testing methodology are as follows:

1. The page flow navigational model is generated through application development.
2. The navigational flow of the application is analyzed to produce states for physical web pages, events for actions and the transitions for the links and forwards.
3. Using the states, transitions and the events in step 2, I have created a state machine and is represented in a State Chart XML. A state chart document is also generated automatically in XML dialect which used as an input for testing.

4. I have written a program which will traverse through various conditions and combinations present in the state model. The program will invoke the application, perform actions and relay the user initiated events(like button click or clicking on hyper link) and serves as an intermediary between the UI and application behavior.
5. With the help of testing tool Selenium2.0[6] , I have also created a test scripts which will validate the navigational flow of the page flow application.
6. The successful execution of test criteria- all pages, all actions, all links and all paths results in test coverage criteria satisfied

The methodology I have presented is limited to strut based framework applications, which follows model-view-controller architecture and have navigational model generated in terms of pages, action, links and forwards during page flow application development.

To illustrate how the testing approach in figure 1.1 is applied to the navigational flow, I have created an online shopping page flow application and have used the application as sample in my paper. The program which I have written (as explained in step 4 and 5) of the testing methodology is specific to the online shopping page flow application.

The paper is organized as follows: The second chapter discusses about the background of page flows, strut based applications, unified modeling language and model based testing. The third chapter describes my online shopping web application based on Java page flows and nested page flows built in Oracle workshop for web logic [7]. The fourth chapter discusses about the background of SCXML, implementation of my online shopping application to construct a state chart using SCXML and the test coverage criteria defined for the page flow application. Chapter five illustrates the generation of test scripts with help of automated testing tool Selenium 2.0[6] and successful execution of all the test coverage criteria defined in the SCXML. Chapter 6 presents concluding remarks.

2. BACKGROUND

This section discusses about background information on Struts based frameworks, Net UI page flows and model based testing. Section 2.1 discusses about the Struts based frameworks and section 2.2 discusses about the Net UI page flows, section 2.3 discusses about modeling the software application before coding with the help of Unified Modeling Language(UML) and section 2.4 discusses about the model based testing approach.

2.1. Struts-Based Frameworks

A framework is "a reusable, 'semi-complete' application that can be specialized to produce custom applications" (Fayad and Schmidt, 1997)[15]. Frameworks provide a skeletal support structure made of software components upon which new software applications can be quickly built and organized. Reusability, whereby generic components can be used in new applications, is a key characteristic[15].

Struts based frameworks(SBF) follows Model-View-Controller(MVC) architecture pattern based on different technologies and the basic construct of SBFs includes Actions, Presentations and a Controller that defines navigation through the system [2] . The MVC pattern has three main components in an application:

1. Model - contains the core of the application functionality and domain knowledge which is defined in the configuration file and provides information to the controller to manipulate the business logic [14].
2. View- responsible for the presentation of the model to the users , and

3. Controller- reacts to the user input by accepting the request from the user, depending on the request and the current state of the model, decides which business logic function to invoke and produce the next view requested to the users [14].

Struts based framework(SBFs) have an underlying navigational model and it aids in the design of web application by using graphical representation for this navigation model. Web application using SBFs will have sets of pages and navigation will occur between the pages. The graphical representation of navigational model will have four main element types:

- Actions- maps an incoming HTTP request to the corresponding method needs to be executed[1][2]
- Forwards- represent a destination to which the controller might be directed to[1][2].
- Link- Logical artifacts that represent the HTTP request from the JSP pages to the Actions[1][2].
- Physical Pages- JSP Pages that handles the User Interface[1][2].

2.2. Page Flows

Network User Interface(NetUI) is a technology that provides an MVC pattern framework built on Apache Struts[3][4] called page flow. Net UI Page flow is a flow-control programming model and is useful to build a well structured Java web application. It also helps to create single web application which can have multiple page flows in it called as modular page flow. This modular page flows can be inserted and reused inside the other flows. The page flows separates the presentation logic through Java Server Pages(JSP) , data processing logic through controls and actions as data are made available through controls and decision logic are accessible through actions[1][4].

The main features of page flows include[1][4]: (1) stateful- the state of the application is stored inside an instance of the controller, (2) modular- a single web application can have multiple page flows within it,(3) nested- A complete page flow can be inserted inside another page flow also called as "nested" page flow. The current page flow can pass the control to the nested page flow and still can retain the state of original page flow or come back to the original page flow when required.

Page flows are built on Apache Struts, as a result many elements found in struts such as Actions, Forwards can be re-used by page flow framework. The four main components of page flows are Action, Forward, Link and Pages. We can also describe page flows as graph representing these four components.

2.3. Unified Modeling Language

Modeling is the designing of the software application before coding. Model driven development helps to simplify the process of designing the components of the system and the relations between the different modules. The model is then converted into code manually or automatically, using tools designed for conversion.

Unified modeling language (UML)[18] is a general purpose modeling language widely used in software development[19] and is evolving under the auspices of the Object Management

Group(OMG)[18] version2.3 . UML helps to specify, visualize and document models of software system which includes structure and design to meet the requirements.

The paper focuses on state machine also known as state chart. The most used formalism is UML state chart initially defined by D. Harel. The Goal of UML state chart is to overcome the limitations of traditional finite state machines while retaining the main benefits. With state chart diagrams behavior of the system can be described in a simple and easy to understand graphic formalism. The UML state chart diagrams are directed graphs in which nodes denote the state and connector denotes the transitions. These transitions are labeled by triggering events and optionally by list of executed actions. Events in most general term can be described as the type of occurrences that affects the system. These events are responsible for state changes. Each state on a state chart diagram can contain multiple internal actions. An action is best described as a task that takes place within a state. Action depends on both state of the system and triggering events. UML state chart has also introduce the new concepts of hierarchical nested states and orthogonal regions and extend the notion of actions.

In my paper, I have used State Chart XML (SCXML)[12] which provide rules to describe state chart model in XML dialect. SCXML is not yet standard but it is a work in progress. It provides general purpose execution environment based on Harel's state chart. SCXML is used to describe the behavior of complex state machines in terms of events, states and transitions and can describe notation such as sub-states, parallel states, synchronization and concurrency in SCXML.

2.4. Model-Based Testing

EL-Far and Whittaker [8][1][2] defines model based testing as an approach in which common testing tasks such as test case generation and evaluation of test results are based on a model of the application under test. A model of software depicts its behavior. Behavior can be described in terms of the input sequences accepted by the system, the actions, conditions, and output logic, or the flow of data through the application's modules and routines [8]. The models used in the model based testing can be divided into two main categories- structural(Static) and behavior(dynamic) and two main levels of abstraction- Platform Independent Models(PIM) and Platform Specific Models(PSM)[1][8][15]. The structural models include control flow graphs, dependency graphs and data flow graphs and represents its source code structure and the behavioral models include state machines, state charts and decision table which describes external or black box behavior [1][2][8] required.

In my paper, I have use one model from both structural and behavioral categories. The application which I have designed is based on page flow model which presents the logical flow of the navigation and has a control within the application through controller. Hence, the page flow model can be categorized as a control flow model and comes under the structural category of model based testing. For behavioral model, I have used state charts and have created the state charts using SCXML to model the behavior of the system. The state chart is further used by the automate tool during testing.

3. ONLINE SHOPPING WEB APPLICATION BASED ON JAVA PAGE FLOW

The third chapter describes my online shopping web application based on Java page flows and nested page flows built in Oracle workshop for web logic [7]. In my paper I will present a methodology to test the online shopping page flow web application using SCXML to create a state chart and Selenium 2.0 for automation testing. During the development of the application a page flow overview will be created which shows the graphical summary of the web application. This page flow overview will show all the pages and actions and the relationship between them through links and forwards. Figure 3.1 shows a generalized page flow overview for illustration.

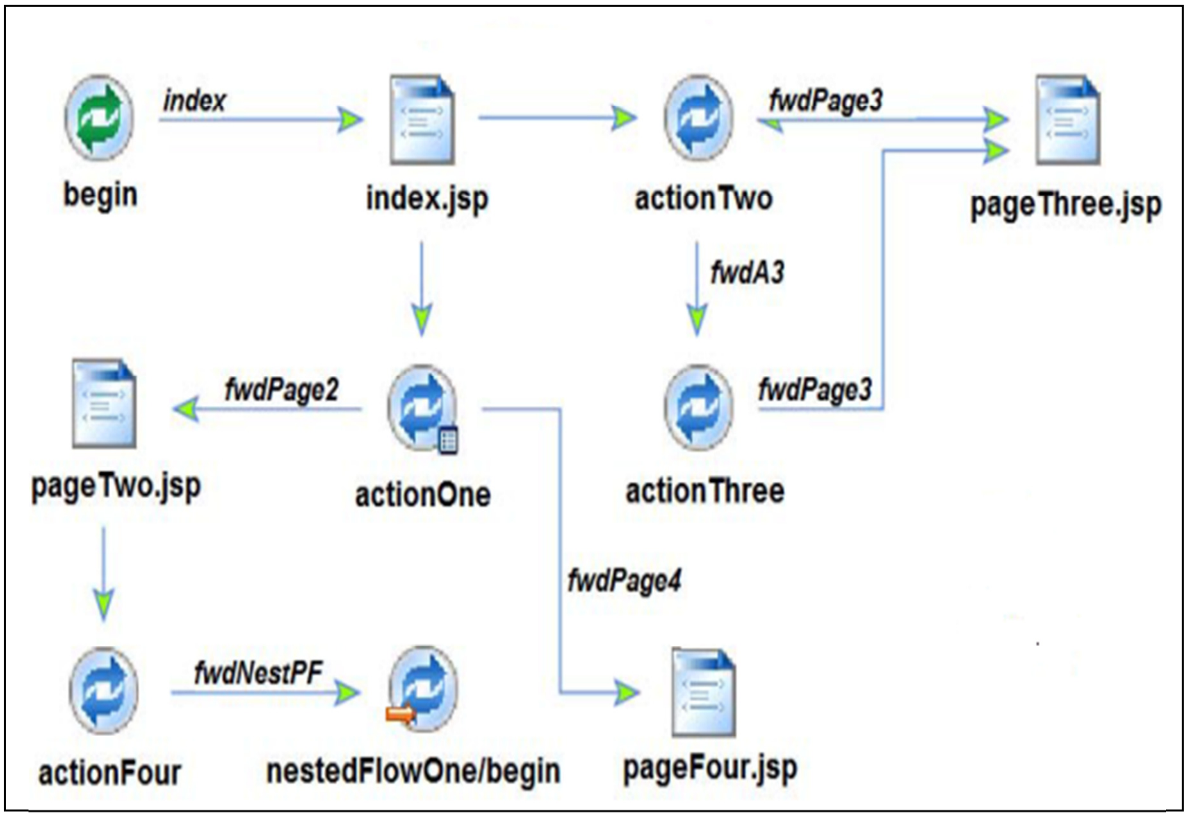


Figure 3.1. Screenshot of Page Flow Overview for Illustrative Example[1]

The page flow application consists of two main types of nodes: Pages and Actions. Navigation between pages are done with help of Actions. Forward and Link transitions are used as a connector between Actions and Pages. Pages, Actions, Forwards and Links are labeled with unique identifiers. The page icon is represented as Pages and has jsp extension. Actions are represented in the circular icon with block arrows. There are four kinds of Actions shown in figure 3.1: (1) Begin Action: Labeled as begin (2) Clean Actions: user defined actions e.g. actionOne (3) Nested flow actions- user defined labels with nested flow one/begin (4) Form Actions: with form icon on circular the circular icon[1]. Links always connect page to an action and are unlabeled transition. Forwards comes out of an Action and are labeled transition.

Figure 3.2 shows the screenshot of my online shopping page flow application which consist of pages, actions, forwards and links. This online shopping application is an e-commerce application that enables customer to purchase goods on the web. The online shopping application consists of navigational flow model shown below in figure 3.2. The elements of this navigational flow model consist of 5 pages, 6forwards, 9 actions and the rest are links with no labels. The Pages include index.jsp, search.jsp, searchResult.jsp, confirm.jsp, complete.jsp. The actions include two simple actions: begin action which is required and search action, six form actions with a form icon: processData, selectItem, confirm, completeOrder, getItem, orderComplete and one nested flow action items/recommendation.jpf. The forwards are the edges exiting the actions: serachResult, confirm, complete, two labeled with "success", and getItemFlow. The links are the edges exiting the pages with no labels which are four in count. One edge from index.jsp, second from search.jsp, third from confirm.jsp and fourth from complete.jsp.

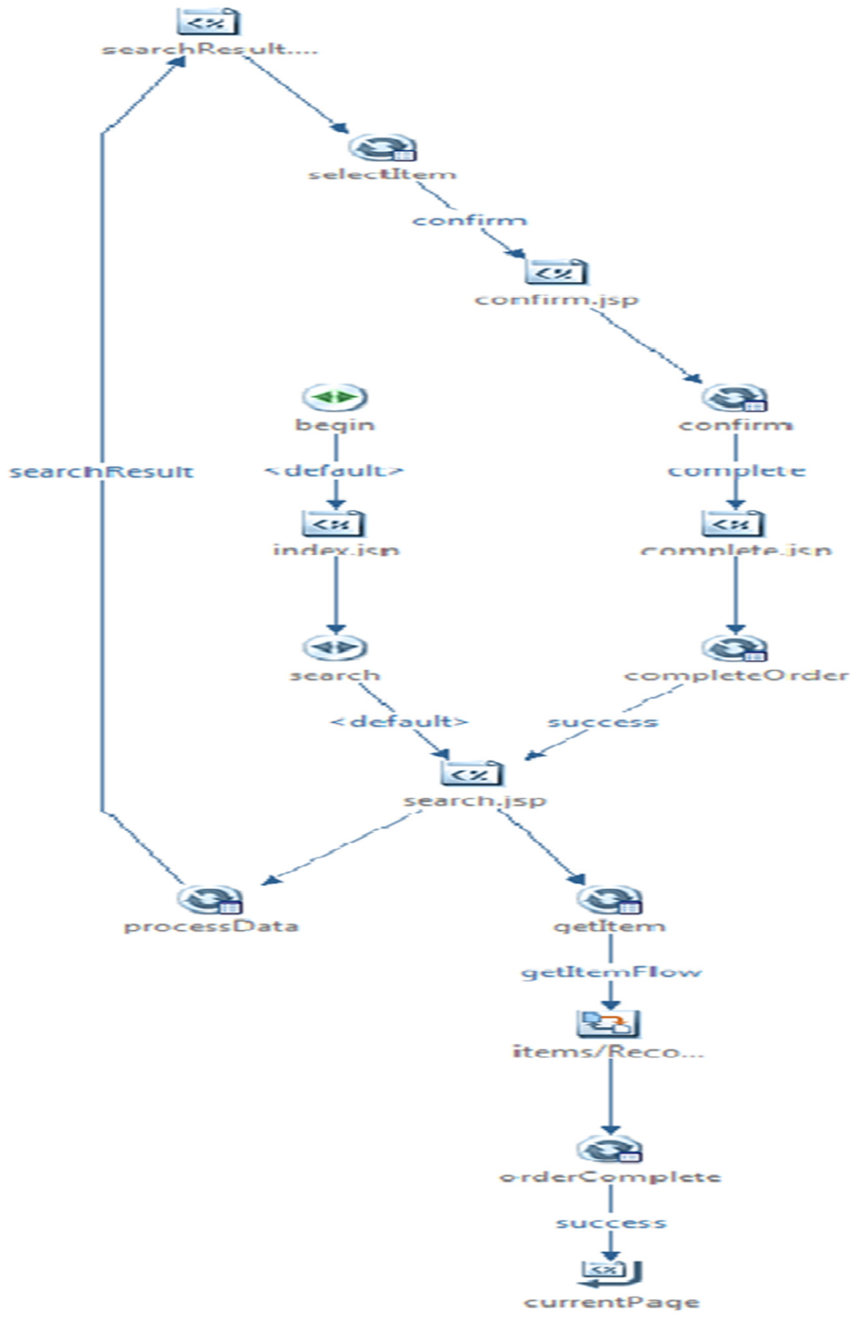


Figure 3.2. Screenshot of Page Flow Overview (Vertical Layout)

3.1. Environment Used to Built the Online Shopping Web Application

The online shopping web based application is designed using java page flow control feature using oracle workshop for web logic [7] where user can search products and place order for the items. Oracle workshop for web logic version 10.3 [7] supports Java enterprise edition 5 and is built on eclipse platform which is an open source framework used for Java development. Below are the software and hardware used to build the online shopping web application:

- **IDE Tool** – Oracle Workshop for web logic developer version which supports multiple web application framework in one integrated development environment: struts, beehive[3][4].
- **SERVER** – Oracle web logic sever 10.3 developer version
- **DATABASE** – Point base database (demo version with the server)
- **LANGUAGE/TECHNOLOGY** – JAVA Page Flow. Page flow is supported by Oracle 10.3 because the Beehive NetUI [3][4] that comes with the server.

3.2. Modules Designed in the Online Shopping Web Application

Page flow is based on strut based framework and follows model view controller user interface design pattern. Web application development using page flow and Java Server Pages (JSPs) helps to separate user interface code, business logic implementation and navigational control. The user interface code can be placed in the JSP files and the implementation of navigational control can be done by page flow's controller file. A controller file is a special java file that uses JPF file extension and is the nerve center of the web application [7].Below is the explanation of different modules designed for online shopping web application using page flow.

3.2.1. Controller Class

A page flow is the Java class called the "controller" class, that controls the behavior of the web application through the use of annotations and methods[7]. These annotation and methods inside the controller class is also known as Actions. Each page flow is a group of files that contains one java controller class and many JSP pages. The users navigate from one page to another with the help of the annotations and methods defined in the controller class. The method and annotations also take care of handling user requests and helps in access of back end resource.

3.2.1.1. Controller.java

In terms of model-view-controller user interface design pattern, the controller.java file is a controller and the JSPs are the view. In every page flow application the begin action must be defined first by the controller class as it is the entry point for the page flow.

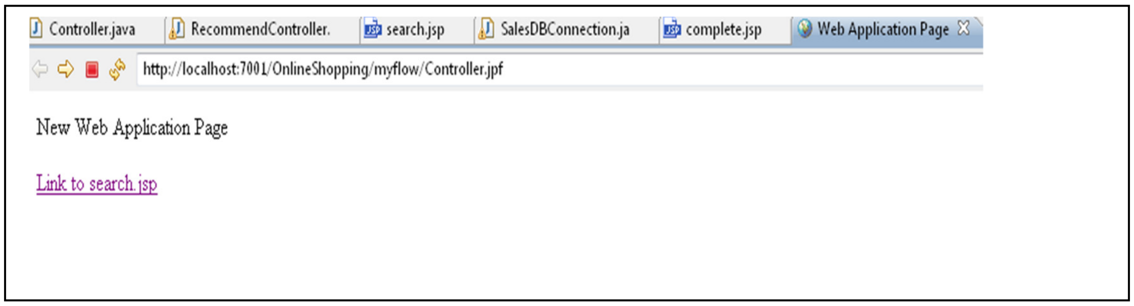
The online shopping page flow application contains two controller class called as Controller.java and RecommendController.java and many number of pages. As mentioned above the controller class will help to navigate from one page to another. Each controller class will have its own form bean. A page flow uses form beans to elicit information from user through form. It is used to encapsulate the data and is defined as a inner class inside the page flow class.

The annotations looks like @annotation-name (parameters) and begins with @. In the annotation @Jpf.controller the navigation of the page is defined. The @Jpf.controller annotation is used for any page flow controller class. This @Jpf.controller annotation alerts the compiler that Controller.java class is the special page flow class instead of typical java class. When a request is received for a page flow controller (controller.java), an action(begin) or a page(index.jsp), an instance of the controller class becomes the current page flow.

The online shopping application starts from controller class "controller.java" where action begin is called first and "index.jsp" page is displayed with a link to search page as shown in figure 3.3.

The "@Jpf.SimpleAction" is used when an action only needs to navigate to a different page without using complex logic. A simple action is implemented to handle navigation, form submission and form validation, it cannot handle decision logic.

```
@Jpf.Controller( simpleActions =
{
@Jpf.SimpleAction(name = "begin", path = "index.jsp"),
@Jpf.SimpleAction(name="toPage2", path="search.jsp")
}
```



New Web Application Page

Link to search.jsp

Figure 3.3. Screenshot of Index Page

3.2.2. Search.jsp

When "Link to search .jsp" is clicked in the above figure 3.3, action "toPage2" is called and will navigate to "Search.jsp" page as shown in above code and "Product Search Page" will be displayed. The below figure 3.4 shows the "Product Search Page".

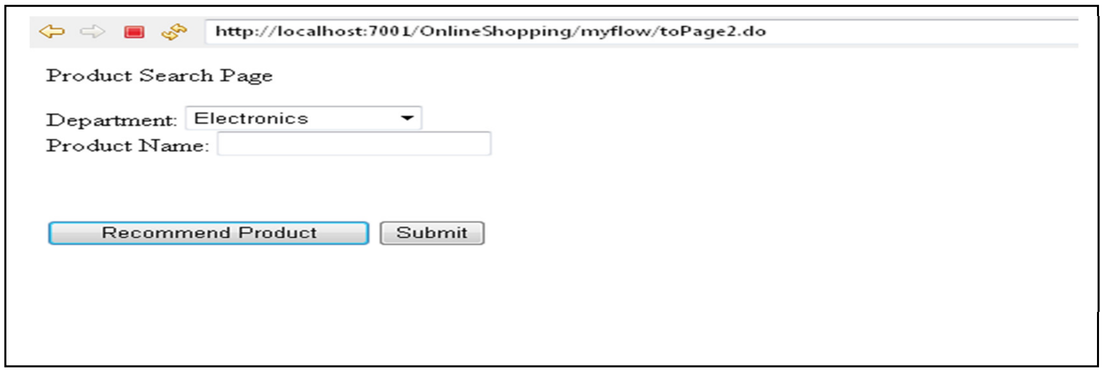


Figure 3.4. Screenshot of Product Search Page

As we can see in the above figure 3.4, Product Search Page has two options

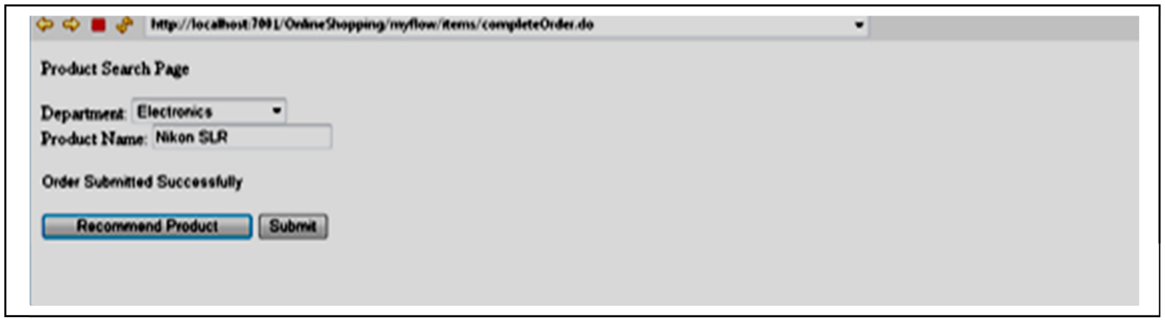
- Recommend Product (nested page flow)
- Submit

We can enter valid information in "Product Name" and hit "Submit" option or we can hit "Recommend Product" option to view recommendations. When "Submit" button is clicked the control is still with "Controller.java" class and a Form Bean is created. When the form data is submitted, the java class will be instantiated and the form data will be loaded into the Java Bean properties of the new instance. In the code below, when Action is created through method, the method will return "forward" and will be annotated through @Jpf.Action. If action needs to make a decision and conditionally execute code based on that decision, an action method is implemented as shown below in the code.

```
Jpf.Action (
                useFormBean="profileForm",
                 forwards={
                @Jpf.Forward(name="searchResult", path="searchResult.jsp")
     },
     doValidation=false
```

In the above code, the action here is "searchResult" and after the processData function is called the action will navigate to "searchResult.jsp" page. The search can be done in two ways

- By selecting only Department and keeping the Product Name empty as shown in the figure 3.4
- By select Department, enter Product Name and hit "Submit" button as shown in the screenshot below in the figure 3.5


http://localhost:7001/OnlineShopping/myflow/items/completeOrder.do

Product Search Page

Department: Electronics

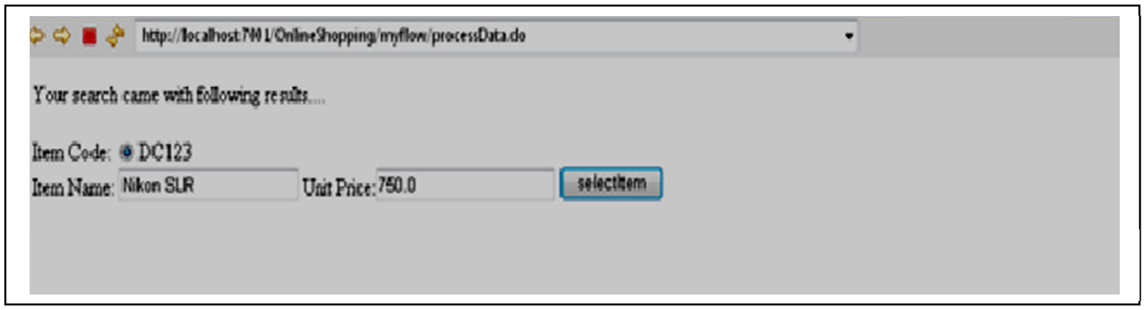
Product Name: Nikon SLR

Order Submitted Successfully

Figure 3.5. Screenshot of Product Search Page with Product Name

3.2.3. SearchResult.jsp

As shown in the above figure 3.4 and figure 3.5, when we enter the valid information in the "Product Search Page" and hit submit button, this page accepts the data send to function "processData" and will display the search results (shown in below screenshot figure 3.6) as per the search information entered. User can select the item to purchase and click on "selectItem" button to proceed further as shown in the figure 3.6 below.


http://localhost:7001/OnlineShopping/myflow/processData.do

Your search came with following results....

Item Code: ● DC123

Item Name: Nikon SLR Unit Price: 750.0 selectItem

Figure 3.6. Screenshot of Search Result Page

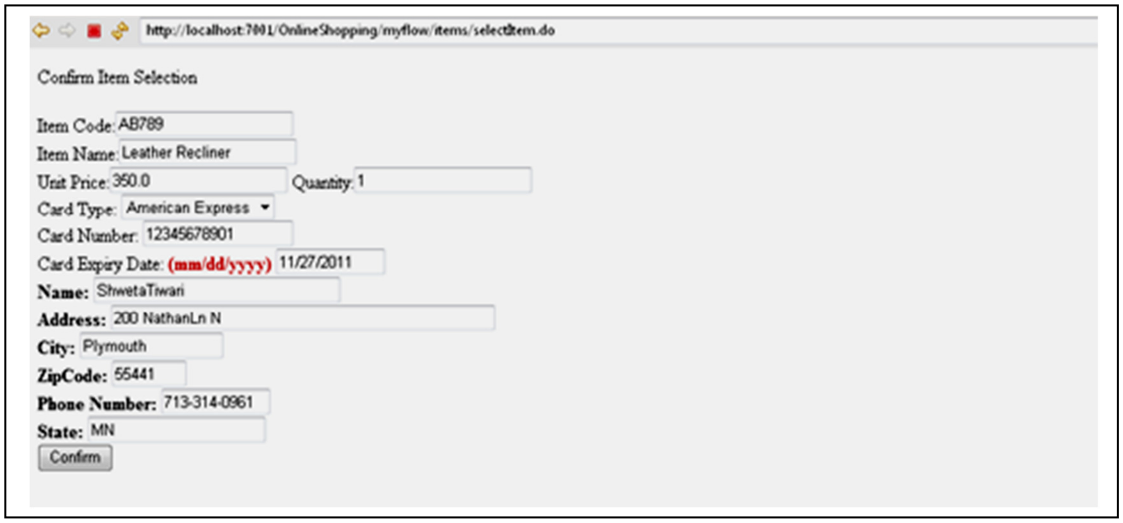
When the user selects the item and hit "selectItem" button, the current information is forwarded to a new page "Confirm.jsp" as shown in figure 3.8 below.

```
@Jpf.Action(
                         useFormBean="itemform",
              forwards = {@Jpf.Forward(name="confirm",
path="confirm.jsp")
              },
              doValidation=false
           )
           public Forward selectItem(ItemForm form) {
              itemName = form.getItem();
              unitPrice= form.getPrice();
              itemCode = form.getCode();
              return new Forward("confirm");
```

Figure 3.7. Code Snippet to Get Data Form and Send Information to "Confirm.jsp" Page

3.2.4. Confirm.jsp

The code snippet above will navigate to "Confirm.jsp" page, once valid information is entered in the "Search Result" page. In the figure 3.8 below, user is required to fill necessary details, review selected items to process the order and hit the "Confirm" button to process the order.


http://localhost:7001/OnlineShopping/myflow/items/selectItem.do

Confirm Item Selection

Item Code:	AB789
Item Name:	Leather Recliner
Unit Price:	350.0
Quantity:	1
Card Type:	American Express
Card Number:	12345678901
Card Expiry Date: **(mm/dd/yyyy)**	11/27/2011
Name:	ShwetaTiwari
Address:	200 NathanLn N
City:	Plymouth
ZipCode:	55441
Phone Number:	713-314-0961
State:	MN

Figure 3.8. Screenshot of Confirm Page

The "Confirm" button when clicked, calls the function "confirm" to get data from the form, calculate the total price and forward it to "Complete.jsp" page as shown in the figure 3.10. Below is the code snippet shown in the figure 3.9

```
@Jpf.Action(
                    useFormBean="confirmform",
      forwards = {
        @Jpf.Forward(name="complete", path="complete.jsp")
      }
)
public Forward confirm(confirmForm form){
    itemName = form.getItem();
    unitPrice= form.getPrice();
    itemCode = form.getCode();
    itemQuantity = form.getQuantity();
    cardType=form.getCardtype();
    expDate= form.getDate();
```

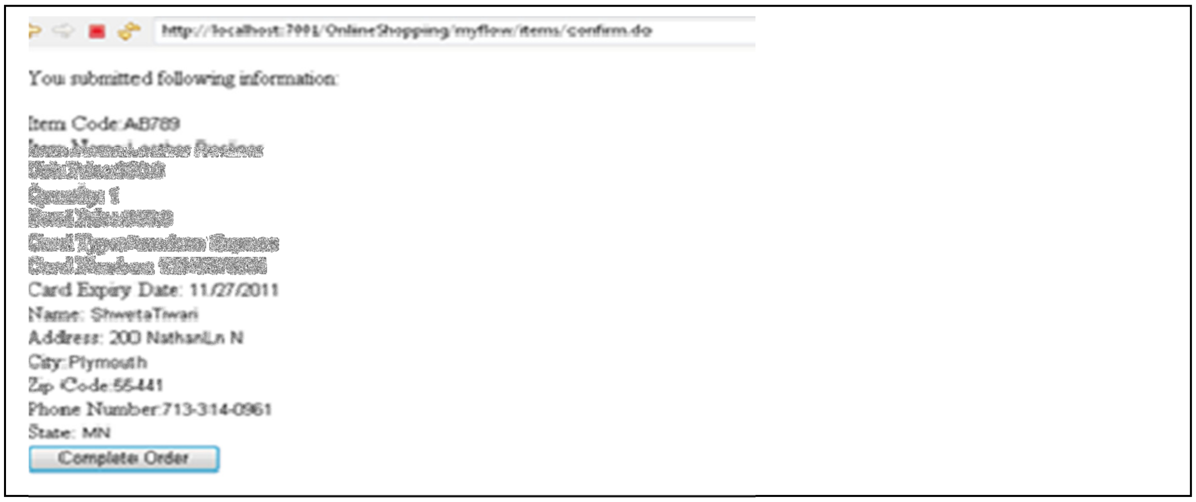
Figure 3.9. Code Snippet to Get Data and Forward Information to "Complete.Jsp" Page

```
        cardNo=form.getCardno();
        custName=form.getName();
        custAddress=form.getAddress();
        custCity=form.getCity();
        custPhone=form.getPhone();
        custState=form.getState();
        zipCode=form.getZip();
        totalprice= unitPrice * itemQuantity;
        return new Forward("complete");
    }
```

Figure 3.9. Code Snippet to Get Data and Forward Information to "Complete.Jsp" Page (Continued)

3.2.5. Complete.jsp

In the "Complete.jsp" page, user is required to review order and click on "Complete Order" button to confirm and process the order. When "Complete Order" button is clicked, "Complete Order" action is called to insert record into the database, send a successful string message and again forward control to "Search.jsp" page for any new search as shown in figure 3.4. The figure 3.10 below shows the screenshot for "Complete.jsp" page.


http://localhost:7001/OnlineShopping/myflow/items/confirm.do

You submitted following information:

Item Code:AB789
[redacted]
Card Expiry Date: 11/27/2011
Name: ShwetaTiwari
Address: 200 NathanLn N
City:Plymouth
Zip Code:55441
Phone Number:713-314-0961
State: MN

Figure 3.10. Screenshot of Complete.jsp Page

The code snippet below in the figure 3.11 describes when the order is reviewed and complete, the forward action is explicitly defined to go back to the search page again.

```
@Jpf.Action(useFormBean="completeform",
        forwards = {
           @Jpf.Forward(
              name="success",path="search.jsp")
        }
```

Figure 3.11. Code Snippet to Go Back to "search.jsp" Page

3.3. Nested Page Flow

As mentioned above in section 3.2.1.1, the online shopping page flow application contains two controller classes called as Controller.java and RecommendController.java. When the user hits on the Submit button on "Product Search Page" as shown in the figure 3.4 ,the control is with controller class controller .java till the order is complete . If the user hits on "Recommend Product" button, controller class "RecommendController.java", defined as "nested page flow" takes the control. The nesting features of page flow helps to enforce modular design and break up a large project into smaller functionalities. With the help of the nesting we can create separate controllers for different section of the project and can temporarily transfer the control to another page flow and can return back to original one when required.

3.3.1. RecommendController.java

Going back to "Product Search Page" in figure 3.4, if we click on "Recommend Product" button, the function "getItem" which is defined in the controller class "Controller.java" is invoked which return new forward "getItemFlow" and initiate the action to pass control to the new controller class "RecommendController.java". The code snippet below in figure 3.11 describes the nested controller class "RecommendController.java", in which the nested page flow takes the control. The @Jpf.controller annotation is used for any page flow controller class, in the below code snippet it is used for nested controller class.

```
@Jpf.Action(
        useFormBean="profileForm",
        forwards={
            @Jpf.Forward(name="getItemFlow", path="items/RecommendController.jpf")
        },
        doValidation=false
    )
    protected Forward getItem(SearchProduct form) {
        return new Forward("getItemFlow");
    }
```

Figure 3.12. Code Snippet "RecommendController.java", the Nested Page Flow

3.3.2. Index.jsp

The nested controller class "RecommendController.java" controller has its own class, actions, form beans and java pages like the main controller class "Controller.java". When the control is passed to this nested page flow controller class "RecommendController.java" an action "begin "is is performed first and navigate to "index.jsp" page. Below is the code snippet for "index.jsp" page in figure 3.13

```
@Jpf.Controller(
   nested = true,
   simpleActions = {
      @Jpf.SimpleAction(name="begin",
path="index.jsp")
   }
```

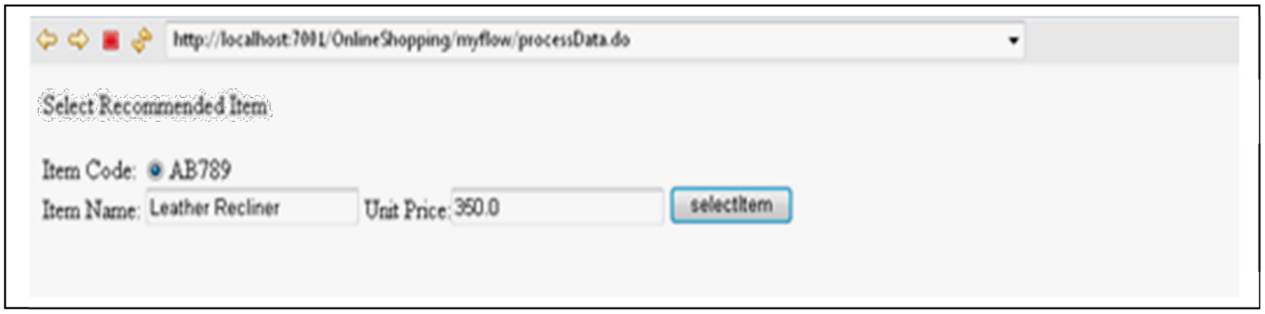
Figure 3.13. Code Snippet for "index.jsp" Page

When program executes first function “getItems” defined in “RecommendController” class above in the code snippet figure 3.12, it retrieves record from the database using defined JDBC control and displays data back to “index.jsp” as shown in the figure below 3.15. Below is the code snippet for retrieving records from database.

```
public Product getItems() throws SQLException{
        String[]prodCode=jdbcCtrl.getProductCode();
                Product items=jdbcCtrl.getProductDetails(prodCode[1]);
                return items;
        }
```

Figure 3.14. Code Snippet for Retrieve Records from Database

In the "index.jsp" page the user selects to purchase and click on “Select Item” button. When clicked on “SelectItem” button, function “selectItem” defined in “RecommedController.java” class controller is called. This function takes data from the form and forward information to next page “confirm.jsp” . Figure 3.15 and 3.16 below shows the screenshot of "index.jsp" page and the code snippet to forward to "confirm.jsp" page.



Select Recommended Item

Item Code: ● AB789
Item Name: Leather Recliner Unit Price: 350.0

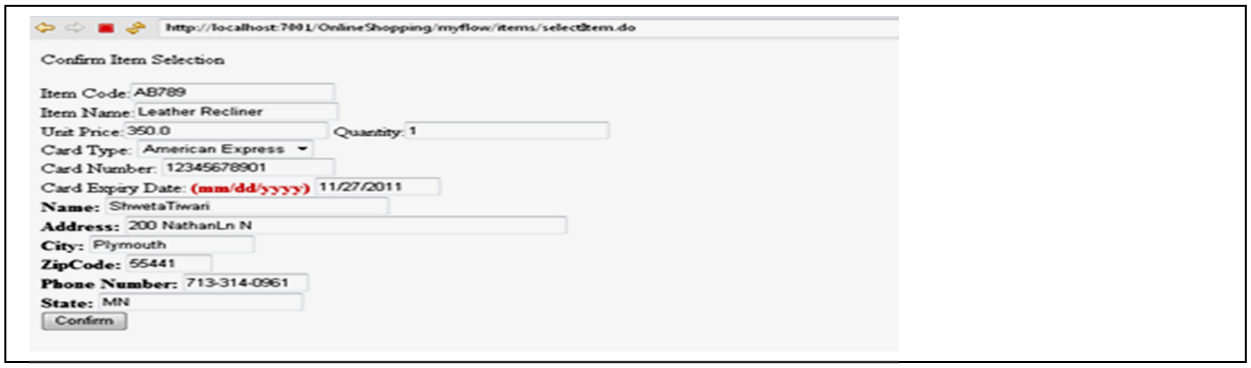
Figure 3.15. Screenshot of Index.jsp Page

```
@Jpf.Action(
        forwards = {
            @Jpf.Forward(name="confirm", path="confirm.jsp")
        }
    )
    public Forward selectItem(ItemForm form) {
        itemName = form.getItem();
        unitPrice= form.getPrice();
        itemCode = form.getCode();
        return new Forward("confirm");
    }
```

Figure 3.16. Code Snippet for "Confirm.jsp" Page

3.3.3. Confirm.jsp

Confirm.jsp takes data from the index page, displays it and wait for user to fill in additional details in order to complete transaction. Figure 3.17 below is the screenshot for "Confirm.jsp" page.


http://localhost:7001/OnlineShopping/myflow/items/selectItem.do

Confirm Item Selection

Item Code: AB789
Item Name: Leather Recliner
Unit Price: 350.0 Quantity: 1
Card Type: American Express
Card Number: 12345678901
Card Expiry Date: (mm/dd/yyyy) 11/27/2011
Name: ShwetaTiwari
Address: 200 NathanLn N
City: Plymouth
ZipCode: 55441
Phone Number: 713-314-0961
State: MN

Figure 3.17. Screenshot of Confirm.jsp Page

When the user clicks on “Confirm” button as shown in above screenshot figure 3.17, “confirm” function is invoked. This function takes data from the form, calculates total price for the order and forward information to “Complete.jsp” page to display the data entered by the user for review.

```
@Jpf.Action(
        forwards = {
            @Jpf.Forward(name="complete", path="complete.jsp")
        }
    )
  public Forward confirm(confirmForm form){
     itemName = form.getItem();
     unitPrice= form.getPrice();
     itemCode = form.getCode();
     itemQuantity = form.getQuantity();
     cardType=form.getCardtype();
      expDate= form.getDate();
      cardNo=form.getCardno();
     custName=form.getName();
     custAddress=form.getAddress();
```

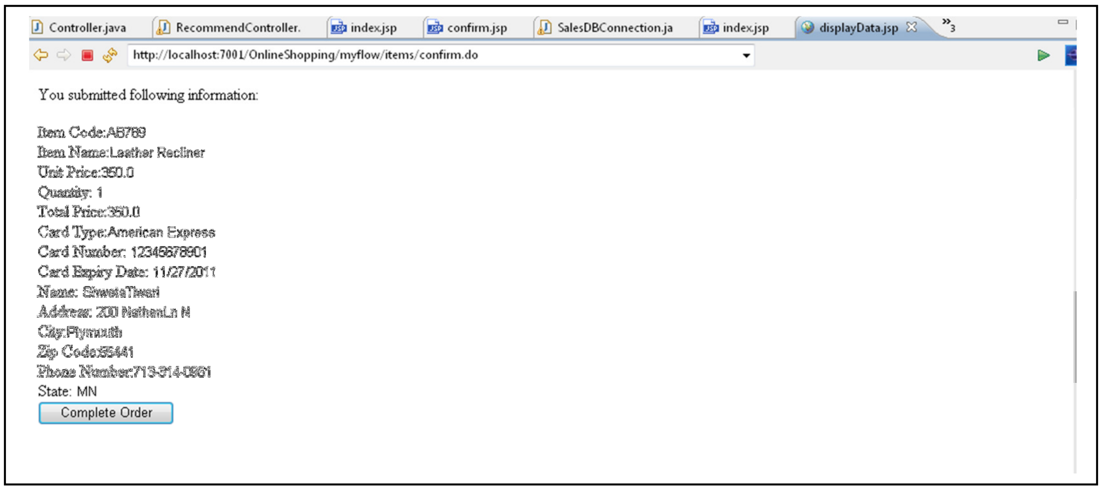
Figure 3.18. Code Snippet to Forward Information to "Complete.jsp" Page

```
        custCity=form.getCity();
        custPhone=form.getPhone();
        custState=form.getState();
        zipCode=form.getZip();
        totalprice= unitPrice * itemQuantity;
        return new Forward("complete");
    }
```

Figure 3.18. Code Snippet to Forward Information to "Complete.jsp" Page (Continued)

3.3.4. Complete Order.jsp

In “complete.jsp” user is required to review order and click on “complete order” button to process order. When clicked on “complete order” button “completeOrder” action is called to insert record into the database, send a successful string message and return back to “search.jsp” page. Figure 3.19 below is the screenshot for "CompleteOrder.jsp" page, Figure 3.20 is the code snippet for "Complete Order.jsp" page and Figure 3.21 is the code snippet which insert record into database and return a new forward "success" along with the string message.



```
You submitted following information:

Item Code:AB789
Item Name:Leather Recliner
Unit Price:350.0
Quantity: 1
Total Price:350.0
Card Type:American Express
Card Number: 12345678901
Card Expiry Date: 11/27/2011
Name: ShwetaTiwari
Address: 200 NathanLn N
City:Plymouth
Zip Code:55441
Phone Number:713-314-0861
State: MN
```

Figure 3.19. Screenshot of Complete Order.jsp Page

```
@Jpf.Action(
        forwards = {
           @Jpf.Forward(
              name="success",
              returnAction="orderComplete",
              outputFormBeanType=String.class)
        }
     )
```

Figure 3.20. Code Snippet for "CompleteOrder.jsp" Page

```
public Forward completeOrder(completeForm form) throws
SQLException {
      itemName = form.getItem();
      unitPrice= form.getPrice();
      itemCode = form.getCode();
      itemQuantity = form.getQuantity();
      cardType=form.getCardtype();
      expDate= form.getDate();
      cardNo=form.getCardno();
      custName=form.getName();
      custAddress=form.getAddress();
      custCity=form.getCity();
      custPhone=form.getPhone();
      custState=form.getState();
      totalprice=form.getTotal();
      zipCode=form.getZip();
jdbcCtrl.insertOrder(itemCode,itemQuantity,totalprice,
custName,cardType,cardNo,expDate,custAddress,custCity,custPh
one, custState,zipCode,unitPrice );
      String message = "Order Submitted Successfully";
      return new Forward("success", message);
}
```

Figure 3.21. Code Snippet to Return a New Forward "success" Along with the String Message

New forward "success" is defined with return action "orderComplete" as shown in the code snippet below in figure 3.22. After order is processed with success, control is returned back to main controller class "Controller.java" where return action "orderComplete" is defined. This action takes the string returned from the nested page flow and updates the field in the form and returns to the original Product Search page shown in figure 3.4. Figure 3.22 below is the code snippet to return to main controller "Controller.java".

```
@Jpf.Action(
        forwards={
            @Jpf.Forward(name="success",
navigateTo=Jpf.NavigateTo.currentPage)
        }
    )
    protected Forward orderComplete(String m) {
            profileForm.setMsg(m);
        Forward success = new Forward("success", profileForm);
        return success;
    }
```

Figure 3.22. Code Snippet to Return to Main Controller "Controller.java"

4. STATE MACHINE REPRESENTATION USING SCXML FOR AUTOMATED TESTING

This chapter discusses about the state-based model of my online shopping application which will represent the behavior of the system under test which can be later used for automated testing. As discussed in chapter 2, the behavioral model used in model based testing includes state machines, sequence diagrams and decision tables. In order to create a state-based model, the tester should have a good knowledge about the components needs to be tested, proper count of inputs and outputs and the target areas needs to be explored.

In my approach, the online shopping web application which consist of actions, pages ,links and forwards is analyzed and using this information states, transitions and events are identified. A state chart is created using state chart xml (SCXML). This state chart will define all-actions, all-pages, all-forwards and all links in which pages will be represented as states, actions as events and forwards and links as transitions. The state chart will also help to generate test information based on the test coverage criteria all-actions, all-pages, all-forwards and all links which is later utilized to test the navigational flow of the online shopping web application and to generate test scripts.

State Chart XML(SCXML) [12] is currently a working draft published by the World Wide Web Consortium(W3C)[12]. SCXML is not yet a standard but it is a work in progress. The latest working draft is dated December 2012 [12]. It provides rules to describe state chart model in a XML dialect. SCXML provides a generic state-machine based execution environment. State in SCXML can be defined as a behavior of the object and transition allows changing the state when events are triggered. Each state contains a set of transition that defines how it reacts with events[12]. The state machine is always in the single state called as "Active State". When an event is triggered, the state machine checks for the transitions defined in the active state and moves to the state specified by the transition, which is also called as "target state". The target state now becomes the active state. The <state> element may also have a "nested" <state> elements and such a state will be called as "Compound State" or "Parent State" and the nested elements as "Child State".

The page flow based online shopping web application is analyzed to produce states for the physical web pages and transition for forwards and links. With the help of page flow overview described in chapter 3 in figure 3.1 (shown below), a state chart is created using SCXMLGUI [9] tool, which is a graphical editor for SCXML[9].

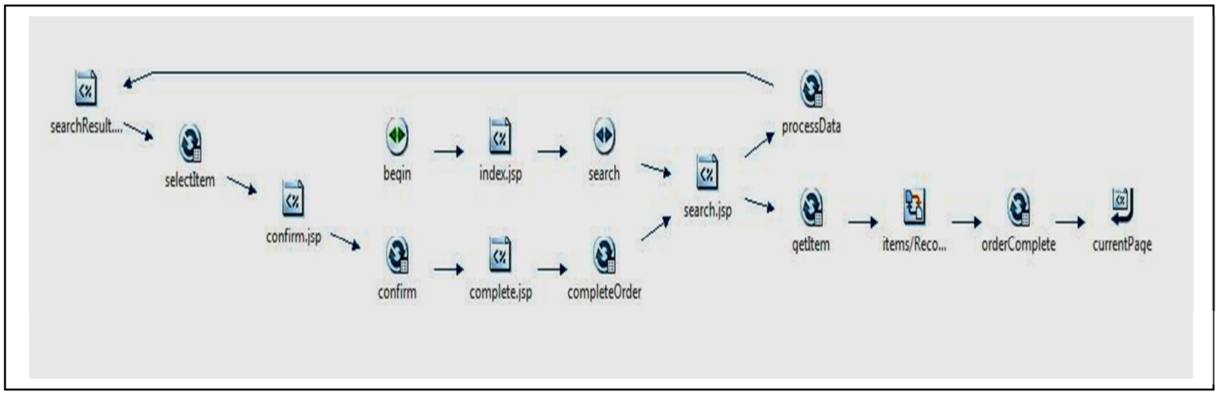


Figure 4.1. Screenshot of Online Shopping Page Flow Application Overview (Horizontal Layout)

The State chart diagram below in figure 4.2 is created and will help to generate test information and help further in analyzing test coverage criteria all-actions, all-pages, all-forwards, all links and the navigational flow of online shopping application.

In the figure 4.2 below, rectangles with the smoothed corners are represented as states and arrows are represented by transition events. As explained in chapter 3, the online shopping application has two controller class- "Controller.java", which is the main controller and "RecommendController.java" , nested page flow takes the control. In the state diagram below in figure 4.2, The Parent <state> element is active and the control is with "Controller.java", the main controller class. The initial state is "index" and the transition event "shopping. search" triggers and allows to move it to the target state "search". The "search" is now the active state and there are two transition events-"shopping.processData", "shopping.getItem". If the transition event "shopping.processData" triggers, the control of the application is still with "controller.java", the main controller class and the parent state elements are active and it allows to move to the target state "search result" which becomes the active state. If the transition event "shopping.getItem" triggers, the child state element becomes active and the control of the application is passed to the nested page flow controller class "RecommendController.java" and the "getItemFlow" becomes the target state. When the parent state elements are active, the "search result" is the active state, transition event "shopping.selectItem" triggers and allows moving to the target state "confirm" and similarly "shopping.confirm" transition event triggers and target state "complete" is achieved. Once the order is successfully completed, transition event "shopping.complete" order triggers and move back to the "search" state as a target state, which becomes the active state again.

When "getItemFlow" becomes the target state and therefore the active state and the control of the application is with the nested page flow controller class "RecommendController.java", the child state elements are active and transaction event" shopping.begin" triggers and "index1" becomes the target state, henceforth as shown in the figure 4.2 with the series of transition events , we achieve target state "confirm1" and "complete1" and at the end "complete1" becomes the active state and when the transition event "shopping.completeOrder" triggers, the dotted lines in grey color with transition event "shopping.completeOrder" indicates the nesting feature of the online shopping page flow application and the control passes from the nested controller to the main controller. When "complete" becomes the active state and the final state, parent state elements gets active, the control of the application goes to the main controller.

This state chart XML below in figure 4.2 covers all pages, all action ,all forwards and all links testing criteria and the complete navigational flow of the online shopping application. The test information extracted from the state diagram will be later useful for automated testing by traversing the entire path which defines the testing coverage criteria and generating test scripts.

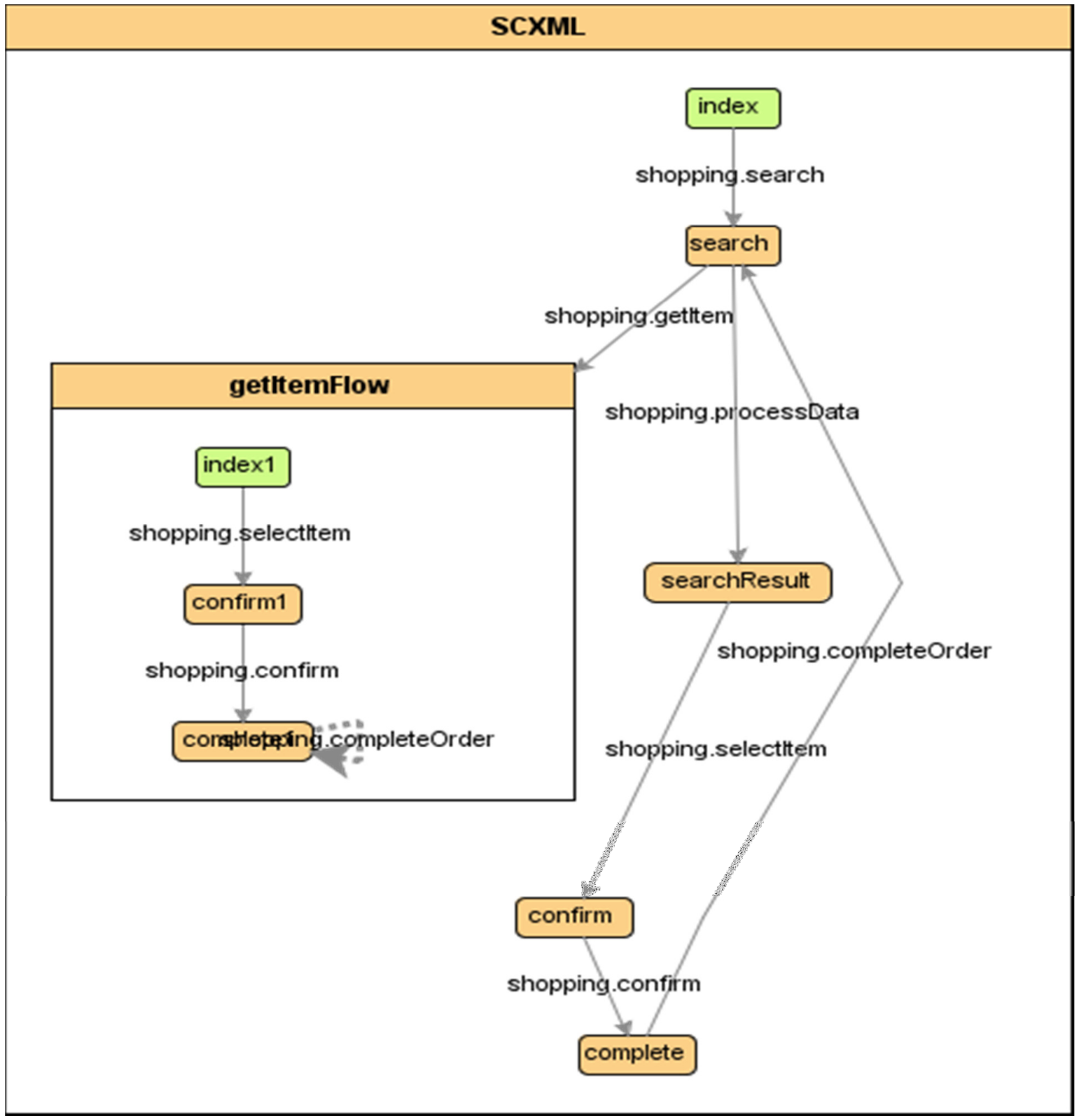


Figure 4.2. State Diagram of Online Shopping Page Flow Application

Figure 4.3 below is the code snippet to create the state diagram of the online shopping page flow application.

```
<scxml initial="index" version="0.9"
xmlns="http://www.w3.org/2005/07/scxml">
<state id="index">
  <transition event="shopping.search" target="search"></transition>
</state>
<state id="search">
  <transition event="shopping.processData"
target="searchResult"></transition>
  <transition event="shopping.getItem"
target="getItemFlow"></transition>
 </state>
 <state id="searchResult">
  <transition event="shopping.selectItem"
target="confirm"></transition>
 </state>
 <state id="getItemFlow" initial="index1">
  <state id="index1">
   <transition event="shopping.selectItem"
target="confirm1"></transition>
  </state>
  <state id="confirm1">
   <transition event="shopping.confirm"
target="complete1"></transition>
  </state>
  <state id="complete1">
   <transition event="shopping.completeOrder"></transition>
  </state>
 </state>
 <state id="confirm">
  <transition event="shopping.confirm"
target="complete"></transition>
 </state>
 <state id="complete">
  <transition event="shopping.completeOrder"
target="search"></transition>
 </state>
</scxml>
```

Figure 4.3. Code Snippet to Create State Diagram of Online Shopping Page Flow Application

5. TESTING METHODOLOGY TO TEST STRUT BASED APPLICATION

The previous chapter 4 discusses about the modeling the strut based application by building a state based model so that it can be further used for automated testing and generating test scripts. Information was extracted from the navigational flow of the online shopping web application and state chart was created to produce model of the application which was utilized for model based testing. An automated testing tool Selenium 2.0 [6] is used in conjunction with the model to generate test scripts and apply them to system under test. Selenium allows to interact with the web browser and almost all the software application are written as a web based applications so that it can run in an internet browser. The main advantage of using Selenium is it supports execution of one's test on multiple browser platforms.

5.1. Testing Approach for Online Shopping Page Flow Application

The steps below describes the testing approach I have used to test the online shopping web application based on strut based framework(SBF).

1. Based on the test criteria all pages, all actions ,all links and all forwards, the navigational flow of the application is analyzed to produce states for physical web pages, events for actions and the transitions for the links and forwards.
2. For state machine representation, I have created a state chart using State Chart XML(SCXML) and is annotated with events on the transitions.
3. I have used automated testing tool Selenium 2.0[6] to write a program which will traverse through various conditions and combinations present in the state model. This testing tool, Selenium, will invoke the application, perform actions and relay the user

initiated events(like button click or clicking on hyper link) to the commons SCXML driven online shopping instance by serving as an intermediary between the UI and application behavior. The program written is limited to my online shopping page flow application and the UI elements needs to be changed for other page flow applications.

4. Finally, automated test case will be generated and this automated test cases will be the selenium test scripts that will validate the navigational flow of my online shopping web application.
5. The successful execution of test criteria- all pages, all actions, all links and all paths results in test coverage criteria satisfied.

5.2. Modeling Inputs

Struts based framework navigational model provides useful information for modeling inputs of a web application. In general, modeling the inputs of an application requires inspection of the GUI to discover the domain of inputs, along with the applicability and behavior constraints on those inputs [10]. For the online shopping web application which is based on strut based framework, the GUI is a set of java server pages that are connected within the navigational flow. This java server pages source code have the information about the components in the GUI like the buttons, text boxes, hyperlinks and it also provides information about nature and type of the possible input values that are valid and can be used. The source code of the actions uses certain inputs and describes the behavior of the application.

5.3. Generation and Execution of Test Script Using Selenium

State chart XML(SCXML) is used to describe the stateful behavior of an objects. The Apache Foundation is supporting SCXML through the Apache Commons project. Commons SCXML[12] is an implementation of a Java SCXML engine capable of executing a state machine defined using a SCXML document as shown in figure 4.3, while abstracting out the environment interfaces[12]. The latest implementation of Commons SCXML is v.0.9. Developers can use the SCXML state model directly in the corresponding code artifacts with the help of Commons SCXML[5].

For the online shopping application I have used Selenium 2(aka Selenium Web driver)[6] which is the newest addition in selenium toolkit and useful for automating web application testing. The figure 5.1 below is the code snippet in which the class "OnlineShoppingSelenium.java" relays user initiated events to the commons SCXML driven "shopping" object which encapsulates the behavior of the online shopping application.

```
public class OnlineShoppingSelenium  {
	private Shopping shopping;


	public OnlineShoppingSelenium(){
		super();
		shopping= new Shopping();
		startApplication();
		shopping.stopAppliction();
	}

		 }

public void startApplication(){
		shopping.fireEvent(Shopping.EVENT_SEARCH);
		testSearchFlow();
```

Figure 5.1. Code Snippet To Relay User Initiated Events

```
testNestedFlow();

        }


       public void testNestedFlow(){
               shopping.fireEvent(Shopping.EVENT_GETITEM);
               shopping.fireEvent(Shopping.EVENT_SELECTITEM);
               shopping.fireEvent(Shopping.EVENT_CONFIRM);
               shopping.fireEvent(Shopping.EVENT_COMPLETEORDER);

       }
public void testSearchFlow(){
               shopping.fireEvent(Shopping.EVENT_PROCESS);
               shopping.fireEvent(Shopping.EVENT_SELECTITEM);
               shopping.fireEvent(Shopping.EVENT_CONFIRM);
               shopping.fireEvent(Shopping.EVENT_COMPLETEORDER);

        }
    public static void main(String[] args) {

             OnlineShoppingSelenium onlineShoppingSelenium = new
OnlineShoppingSelenium();
       }
 }
```

Figure 5.1. Code Snippet to Relay User Initiated Events (Continued)

The Class "Shopping.java" is the selenium test script shown in figure 5.3 and implements the online shopping application behavior which is based on the strut based framework. In the code snippet below the class "Shopping.java" extends "AbstractStateMachine" which provides the base functionality needed by the classes representing stateful entities, whose behaviors are defined via SCXML documents shown in figure 4.2 [12]. The events which needs to be processed are also shown below in the code snippet.

```
public static final String
EVENT_SEARCH = "shopping.search",
EVENT_PROCESS = "shopping.processData",
EVENT_GETITEM = "shopping.getItem",
EVENT_SELECTITEM = "shopping.selectItem",
EVENT_CONFIRM = "shopping.confirm",
EVENT_COMPLETEORDER = "shopping.completeOrder";
```

In the code snippet below, the information is pulled directly from the behavioral model of the online shopping application which is encapsulated in the SCXML document "OnlineShoppingStateChart.scxml" shown in figure 4.2.

```
super(Shopping.class.getClassLoader().getResource("OnlineShoppingStateChart.scxml"
));
```

For my application the browser and test both runs on the same machine, I am using web driver API and not the selenium server. The test will use the web driver API and web driver will run the browser directly. For my application I am using Internet Explorer to run the test and new instance of internet explorer is created in the test script and the get method will help the web driver to navigate to the JSP page shown in the code snippet below.

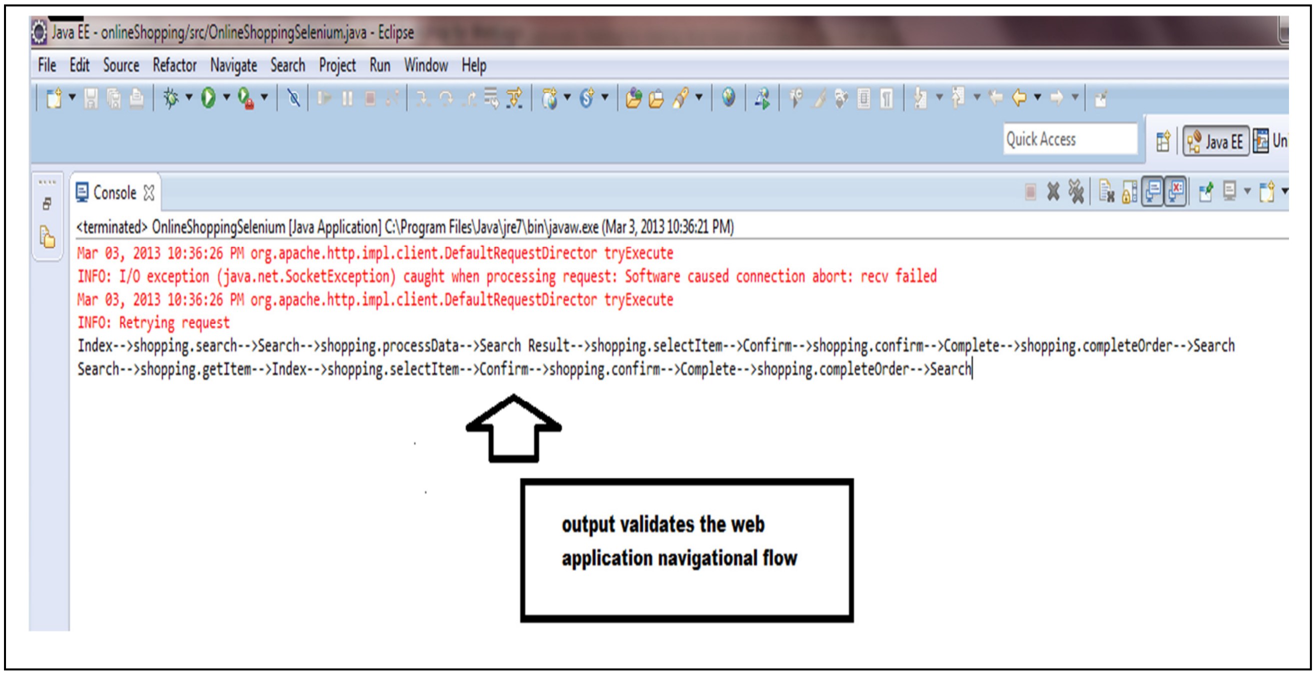
```
driver = new InternetExplorerDriver();
driver.get("http://localhost:7001/OnlineShopping/myflow/Controller.jpf");
```

Selenium uses Locators to find and match the elements of the page which it interacts with. Locating elements in Web Driver can be done on the Web Driver instance itself or on a Web Element[6]. Each of the language bindings expose a “Find Element” and “Find Elements” method[6]. In the selenium test script for my online shopping application, shown below in figure 5.3, "by link text" ,"by id" and by "x-path" locators are used for locating UI elements or web elements.

Each methods in the test script below index(), index1(), search(), getItemflow(), confirm(), complete()) is the activity corresponding to the states in the SCXML document.

5.4. Output

Once we run the selenium test script shown in figure 5.3 , the output displayed in the console conforming the validation of the online shopping web application navigational flow as shown in screenshot figure 5.2. and the test criteria, all pages which was represented as states, all forwards/ links represented by transitions and actions performed as events in the SCXML diagram was covered successfully.


Java EE - onlineShopping/src/OnlineShoppingSelenium.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Console
output validates the web application navigational flow

```
<terminated> OnlineShoppingSelenium [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Mar 3, 2013 10:36:21 PM)
Mar 03, 2013 10:36:26 PM org.apache.http.impl.client.DefaultRequestDirector tryExecute
INFO: I/O exception (java.net.SocketException) caught when processing request: Software caused connection abort: recv failed
Mar 03, 2013 10:36:26 PM org.apache.http.impl.client.DefaultRequestDirector tryExecute
INFO: Retrying request
Index-->shopping.search-->Search-->shopping.processData-->Search Result-->shopping.selectItem-->Confirm-->shopping.confirm-->Complete-->shopping.completeOrder-->Search
Search-->shopping.getItem-->Index-->shopping.selectItem-->Confirm-->shopping.confirm-->Complete-->shopping.completeOrder-->Search
```

Figure 5.2. Screenshot of Selenium Test Script's Output for Online Shopping Application

```
import org.apache.commons.scxml.env.AbstractStateMachine;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.ie.*;
import org.openqa.selenium.support.ui.ExpectedCondition;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.Select;
import org.openqa.selenium.support.ui.WebDriverWait;

public class Shopping extends AbstractStateMachine{

	public static final String EVENT_SEARCH = "shopping.search",
	EVENT_PROCESS = "shopping.processData", EVENT_GETITEM = "shopping.getItem",
	EVENT_SELECTITEM = "shopping.selectItem", EVENT_CONFIRM =
"shopping.confirm",	EVENT_COMPLETEORDER = "shopping.completeOrder";

	public WebDriver driver;
	public WebElement element;
	 private String itemName;
	private Float unitPrice;
	private String itemCode;
	private Integer itemQuantity;
	private String cardType;
	private String expDate;
	private String cardNo;
	private String custName;
	private String custAddress;
	private String custCity;
	private String custPhone;
	private String custState;
	private Float totalprice;
	private String zipCode;
  public Shopping() {
	super(Shopping.class.getClassLoader().getResource("OnlineShoppingStateChart.
scxml"));
		}
	public void index(){

	itemName=itemCode=cardType=expDate=cardNo=custName=custAddress=custCity=cust
Phone=custState=zipCode="";
		totalprice=unitPrice=(float) 0.0;
		itemQuantity=0;
		driver = new InternetExplorerDriver();

	driver.get("http://localhost:7001/OnlineShopping/myflow/Controller.jpf");
		listStates();
	}
```

Figure 5.3. Selenium Test Script for Online Shopping Application

```
	}

	public void getItemFlow(){
	System.out.println();
	listStates();
	element=driver.findElement(By.xpath("//input[@id='rcmnd' and
not(@disabled)]"));
	element.click();
	listEvents(EVENT_GETITEM);
	}

	public void confirm1(){
	listStates();
	element=driver.findElement(By.xpath("//input[@value='A002']"));
	element.click();
	element=driver.findElement(By.xpath("//input[@id='selitem' and
not(@disabled)]"));
	element.click();
	listEvents(EVENT_SELECTITEM);
	}

	public void complete1(){
		if (driver.getTitle().equals("Confirm")){
		listStates();
		element=driver.findElement(By.id("qty"));
		element.sendKeys("1");
		element=driver.findElement(By.id("crdno"));
		element.sendKeys("6781083905756242");
		element=driver.findElement(By.id("date"));
		element.clear();
		element.sendKeys("03/02/2012");
		element=driver.findElement(By.id("name"));
		element.sendKeys("abhishek misra");
		element=driver.findElement(By.id("addr"));
		element.sendKeys("12055 bedford plz");
		element=driver.findElement(By.id("city"));
		element.sendKeys("omaha");
		element=driver.findElement(By.id("zip"));
		element.sendKeys("68164");
		element=driver.findElement(By.id("phno"));
		element.sendKeys("9528319693");
		element=driver.findElement(By.id("state"));
		element.sendKeys("nebraska");
		element=driver.findElement(By.xpath("//input[@id='cnfrm' and
not(@disabled)]"));
		listEvents(EVENT_CONFIRM);
		}
```

Figure 5.3. Selenium Test Script for Online Shopping Application (Continued)

```
if (driver.getTitle().equals("Complete")){
                        listStates();
                        element=driver.findElement(By.xpath("//input[@id='cmplt' and 
not(@disabled)]"));
                        element.click();
                        listEvents(EVENT_COMPLETEORDER);
                        listStates();
                }
                }

        public void searchResult(){
                listStates();
                element=driver.findElement(By.xpath("//input[@id='sbmt' and 
not(@disabled)]"));
                element.click();
                listEvents(EVENT_PROCESS);
        }
        public void confirm (){
                listStates();
                element=driver.findElement(By.xpath("//input[@value='A004']"));
                element.click();
                element=driver.findElement(By.xpath("//input[@id='selitem' and 
not(@disabled)]"));
                element.click();
                listEvents(EVENT_SELECTITEM);
        }

        public void complete(){
                listStates();
                element=driver.findElement(By.id("qty"));
                element.sendKeys("2");
                element=driver.findElement(By.id("crdno"));
                element.sendKeys("6781083905756242");
                element=driver.findElement(By.id("date"));
                element.clear();
                element.sendKeys("03/02/2012");
                element=driver.findElement(By.id("name"));
                element.sendKeys("shweta tiwari");
                element=driver.findElement(By.id("addr"));
                element.sendKeys("12055 bedford plz");
                element=driver.findElement(By.id("city"));
                element.sendKeys("omaha");
                element=driver.findElement(By.id("zip"));
                element.sendKeys("68164");
```

Figure 5.3. Selenium Test Script for Online Shopping Application (Continued)

```
element=driver.findElement(By.id("phno"));
                element.sendKeys("9528319693");
                element=driver.findElement(By.id("state"));
                element.sendKeys("nebraska");
}

element=driver.findElement(By.xpath("//input[@id='cnfrm' and
not(@disabled)]"));
                element.click();
                listEvents(EVENT_CONFIRM);
        }

        public void stopAppliction(){
                driver.close();
        }

        public void listStates(){
                String State= driver.getTitle();
                System.out.print(State);

        }
        public void listEvents(String Event){
                System.out.print("-->"+Event +"-->");

        }
}
```

Figure 5.3. Selenium Test Script for Online Shopping Application (Continued)

6. CONCLUSION AND FUTURE WORK

In this paper I have extend an approach to reduce the manual effort by creation of state-based model and usage of automated testing method for model based testing which describes the behavior of the application based on strut based framework taking into account that It also covers all the test criteria defined which are extracted from the navigational flow of the application. I am also using an existing automated tool Selenium[2] in my example, which is widely used now-a-days for testing web based application in a much simpler way and support multiple platform and operating system. It supports successful execution of test on multiple browser platforms like Internet Explorer, Safari, Firefox, Opera and many more. Selenium also supports object oriented programming language and it allows user to use a wide range and variety of programming language like Java, C#, Ruby, Python, pearl & IDEs like Eclipse, Net beans, Visual Studio etc.

The proposed methodology is implemented on a real time application as an example to show the feasibility of my approach .I have used automated testing method and with the help of automated testing tool Selenium, test scripts are generated which covers all the test criteria defined. Based on this online shopping application, I have also build a state model using SCXML- which is all about state chart and is a powerful extension of finite state machines which describes the flow of the complex web based application in an easy way. It is a general-purpose event-based state machine language and is widely used. SCXML can describe any complex state machine using XML based markup language. It describes the behavior of the application under test in terms of states, events and transition. The automated tool selenium takes the input in terms of elements associated from this state model to test the navigational flow of the application and the test criteria defined.

Future work includes investigation of alternative test case generation technique and investigation of ways to further reduce manual effort required to create state based model for complex web based application. A generic program, software or interpreter can be built which can automatically represent any page flow application in terms of states, transition and events and can create a state based model for the application.

REFERENCES

1. Alava, J., King, T.M., Clarke, P.J. ,Automatic validation of java page flows using model-based coverage criteria, In: Proceedings of the 30th Annual International Computer Software and Applications Conference
2. Alava, J. ,Software-practice and expirence.2000, An automatic method for validating strut based applications, Copyright c 2000 John Wiley & Sons, Ltd.
3. The apache software foundation :http://www.apache.org/Retrieved on 03-04-2013.
4. The apache beehive project: http://beehive.apache.org/docs/1.0.2/netui/overview.html, Retrieved on 03-04-2013
5. The apache commons SCXML. State chart XML(SCXML) 2010: http://commons.apache.org/scxml/ ,Retrieved on 03-04-2013
6. Selenium HQ 2.0, web application testing system: http://seleniumhq.org/ ,Retrieved on 03-04-2013
7. Oracle workshop for web logic 10.3:http://docs.oracle.com/cd/E13224_01/wlw/docs103/ , Retrieved on 03-04-2013
8. Far, I. K. El. and Whittaker, J. A. , Model-based software testing. Encyclopedia of Software Engineering,2001. ed. John J. Marciniak.
9. SCXML GUI editor for SCXML: http://code.google.com/p/scxmlgui/ , Retrieved on 03-04-2013
10. Whittaker, J.A. , Stochastic software testing , Annals of Software Engineering 4 (1997) 115–131

11. Andrews, A.A., Offutt J., and Alexander R. T. Testing web applications by modeling with fsms, Software and Systems Modeling, 4(3):326–345, July 2005.
12. State Chart XML(SCXML): State Machine Notation for Control Abstraction
13. W3C Working Draft 6 December 2012: http://www.w3.org/TR/scxml, Retrieved on 03-04-2013
14. Ricca, F. and Tonella, P. , Analysis and testing of web applications, In Proceedings of the 23rd International Conference on Software Engineering, IEEE, May 2001.
15. Davis, M.: Struts, an open-source MVC implementation (2001), http://www-128.ibm.com/developerworks/ibm/library/j-struts/, Retrieved on 03-04-2013
16. Casal, D.P. , Advanced software development for web applications. Technical Report TSW0505: http://www.jisc.ac.uk/uploaded_documents/jisctsw_05_03pdf.pdf, Retrieved on 03-04-2013
17. Mens, T., Czarnecki, K., Gorp, P.V. , 04101 discussion – a taxonomy of model transformation
18. Object Management Group (2010), http://www.omg.org/spec/UML/2.3/ , Retrieved on 03-29-2013
19. Migliorini, N. ,Comparing UML, SCXML and the Apache implementation of SCXML engine