An Overview of J2EE Architecture and MVC Design Pattern

Deepak Singh
KIET Group of Institutions, Ghaziabad
deepak.singh.mca@gmail.com

Naresh Chandra
KIET Group of Institutions, Ghaziabad
naresh.chandra@gmail.com

Abstract—Java platform, Enterprise Edition (Java EE), is designed to provide the facility to develop distributed, robust, portable and multi-tier application. Enterprise applications must be designed, developed keeping in mind that it should involve lest cost with greater efficiency and with minimum resources. J2EE architecture provides us a standard to develop enterprise applications. It has various parts in the form of component, service and communication. Model-View-Controller is widely adopted design pattern provides an approach to solve real world problem. MVC design pattern is used across many languages and it provides the separation of tasks in three components that is used to develop web applications.

Keywords- J2EE, MVC, EJB.

I. INTRODUCTION

J2EE specifications say nothing about how to implement the specification or how to implement the environment. The components are written in such a way that container can understand or handle it. There are three parts of the J2EE architecture framework.

1) Components: J2EE application framework consists of three kinds of modules: Enterprise Beans components, Web components, and application clients, i.e. JSP, Servlet, EJB.

2) Services: There are various services provided for your component, like: JDBC, JMS, Security and transaction.

3) Communication: This part describes how to communicate between components and services, like: RMI, IIOP etc.

II. J2EE ARCHITECTURE

This diagram shows three tier architecture of J2EE. Client tier contains web client, Applet and any application client. Middle layer has containers like: web container, EJB container and Application Client container. The backend layer contains databases, message service, Naming registry or any other Data Source. One component can communicate with other component by the help of service. Client tier can communicate with middleware using HTTP/HTTPS, RMI or IIOP [6]. Likewise middle tier can communicate with backend layer using Java Messaging Service, JNDI, and JDBC.

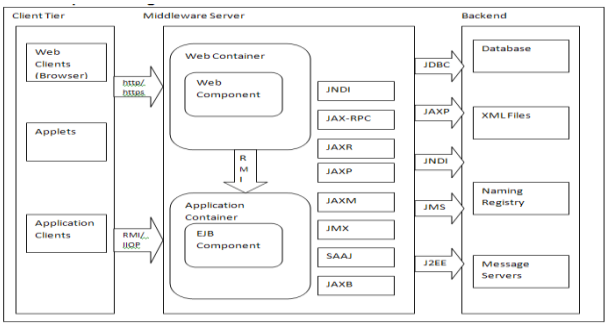

Fig 1. J2EE Architecture [3].

III. CONTAINER ARCHITECTURE FOR J2EE

J2EE provides container architecture for handling various types of component, managing life cycle of the components, providing services to other components and communication between components. Containers are the software components which are capable to manage that type of components. For example web containers are designed to manage Web components. EJB container is capable to manage EJB components. Following are the building blocks of container architecture for J2EE [7]:

1) *Component Contract:* The component should follow certain rules so that containers can certain services for that component [1].
2) *Declarative Services:* All the services (JDBC, JMS, and Security etc.) are made available by the container for the components. Components to tell the container which of these services are required. This is done by deployment descriptor. So declarative services come in picture when

you declare it in deployment descriptor, then that service will be provided for that component.

3) *Container Services:* All the services managed by the container like: JDBC Service, Security, JMS, Transaction etc.
4) *Other Services:* Other services that are directly managed by the container for your component, ex. instance pooling is done by the container, instance allocation.

IV. MODEL-VIEW-CONTROLLER

Design pattern is used to develop web applications with minimum effort and in less time. MVC is a design pattern used to solve a problem related to Object Oriented programming, So OOP and MVC become inseparable. It has following three parts [9]:

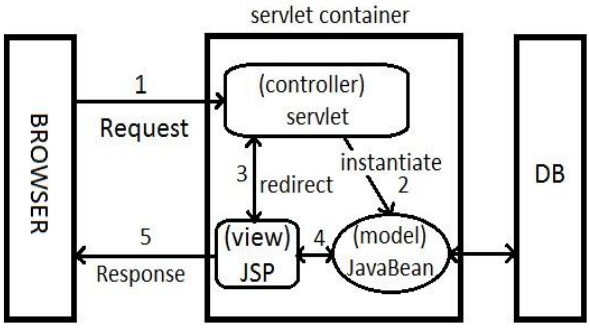

Figure 1: Model-View-Controller [2].

1) *Model:* - The model represents data and the rules that govern access to and updates of this data. In enterprise software, a model often serves as a software approximation of a real-world process. *[1]* Model contains business logic and methods that work on business data. It gives the information to view part of MVC. Model represents a java object that carries data. It also has logic to update the controller if its data changes [10]. If there is an entity Student and we represent it as a Model then its attributes are declared inside the class and there method that can access the data. These get()/set() methods are used to access the data.

```
public class Student{
private String rollNo;
private String name;
public String getRollNo() {
    return rollNo;
  }
public void setRollNo(String rollNo) {
    this.rollNo = rollNo;
  }
public String getName() {
    return name;
  }
public void setName(String name) {
    this.name = name;
  }
}
}
```

2) *View:* The view renders the contents of a model. It specifies exactly how the model data should be presented. If the model data changes, the view must update its presentation as needed. This can be achieved by using a push model, in which the view registers itself with the model for change notifications, or a pull model, in which the view is responsible for calling the model when it needs to retrieve the most current data [1]. The user interacts with application view. In the reference of web application, view technology is used to render user interfaces. Mostly JSP and HTML with CSS is used as view technology.

```
login-success.jsp
<%@page import="com.Student"%>
<p>You are successfully logged in!</p>
<%
 Student                                    bean=
(Student)request.getAttribute("bean");
 out.print("Welcome, "+bean.getName());
 %>
```

3) *Controller:* The controller translates the user's interactions with the view into actions that the model will perform. In a stand-alone GUI client, user interactions could be button clicks or menu selections, whereas in an enterprise web application, they appear as GET and POST HTTP requests [1]. Controller has certain responsibilities:
   a) To manage workflow of the application
   b) It parses the user request.
   c) It identifies URL pattern, form parameters, form method
   d) Selects the next view component to display.

There are various technologies that provide control layer in application framework like: Struts, Spring, Java Server Faces and WebWork [4].

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;
public class ControllerServlet extends HttpServlet {
protected void doPost(HttpServletRequest request,
```

```
HttpServletResponse response)        throws
ServletException, IOException {
 response.setContentType("text/html");
PrintWriter out=response.getWriter();
 String name=request.getParameter("name");
Student bean=new Student();
bean.setName(name);
request.setAttribute("bean",bean);
RequestDispatcher
rd=request.getRequestDispatcher("login-success.jsp");
  rd.forward(request, response);
      }
      }
```

V. INTERACTION BETWEEN MVC COMPONENTS

In the context of an application once the model, view, and controller objects are instantiated, the following occurs:

1. The view registers as a listener on the model. Any changes to the underlying data of the model immediately result in a broadcast change notification, which the view receives. This is an example of the push model described earlier. Note that the model is not aware of the view or the controller -- it simply broadcasts change notifications to all interested listeners.
2. The controller is bound to the view. This typically means that any user actions that are performed on the view will invoke a registered listener method in the controller class.
3. The controller is given a reference to the underlying model [1].

Once a user interacts with the view, the following actions occur [1]:

1. The view recognizes that a GUI action -- for example, pushing a button or dragging a scroll bar -- has occurred, using a listener method that is registered to be called when such an action occurs.
2. The view calls the appropriate method on the controller.
3. The controller accesses the model, updating it in a way appropriate to the user's action.
4. If the model has been altered, it notifies interested listeners, such as the view, of the change. In some architecture, the controller may be responsible for updating the view.

VI. THE DEPLOYMENT DESCRIPTOR: WEB.XML

A web application's deployment descriptor describes the classes, resources and configuration of the application and how the web server uses them to serve the user's requests [5]. When the web server receives a request for the application, it reads the deployment descriptor to map the URL of the request to the code that ought to handle the request [11].

Here is a simple web.xml example that maps all URL paths (/*) to the servlet class :
com.ControllerServlet

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
version="2.5">
   <servlet>
      <servlet-name>comingsoon</servlet-name>
      <servlet-class>com.ControllerServlet  </servlet-
class>
   </servlet>
   <servlet-mapping>
      <servlet-name>myservlet</servlet-name>
      <url-pattern>/*</url-pattern>
   </servlet-mapping>
</web-app>
```

You can declare multiple Servlets using the same class with different initialization parameters. The name for each Servlet must be unique across the deployment descriptor.

VII. CONCLUSION

MVC is the best design pattern for any GUI programmer's. This article has shown how to implement a variation of the MVC design using JSP and Servlet and libraries. MVC provides the facility to multiple data presentations for the same data. Using MVC we can separation of task between components. Web applications developed using MVC is easy to maintain.

REFERENCES

[1] http://www.oracle.com/technetwork/articles/javase/ index-142890.html#1
[2] http://sushantsharmaa.blogspot.in/2012/09/mvc-architecture.html
[3] http://tekmarathon.com/2012/10/09/introduction-to-j2ee-2-x-its-architecture-and-component-diagram/
[4] www.utdallas.edu/chung/patterns/1.DesignPatterns .ppt
[5] https://developers.google.com/appengine/docs/java/config/ webxml
[6]http://docs.oracle.com/javaee/7/tutorial/doc/home.htm
[7]www.it.iitb.ac.in/~kaushal/downloads/oos_project_report.pdf
[8]http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
[9] http://www.tutorialspoint.com/design_pattern/
[10]http://www.journaldev.com/1827/java-design-patterns-example-tutorial
[11] P.Gupta, M.C.Govil, MVC Design Pattern for the multi framework distributed applications using XML, spring and struts framework, International Journal on Computer Science and Engineering Vol. 02, No. 04, PP. 1047-1051, 2010.