Architecting Thick Client Model-View-Controller (MVC) for Web Application

Nitesh Kumar*
Syed Imtiyaz Hassan*

Abstract

Model-View-Controller (MVC) pattern has already been accepted and employed in many successful traditional web applications in recent years. With the growth of social networking web sites and emergence of thick client web application, the issues of usability and response time is more needed than ever. The traditional definition of MVC now misfits itself in such kind of web applications. With the availability of comprehensive Java Script libraries that provides an opportunity to develop thick client web application also forces the developer to look deeper into the new challenges. One among the list may be the standardization of client side MVC and its integration with server side MVC. The present work is an attempt to address the said issue and try to standardize the thick client MVC that can be fit into thick client Web 2.0 application. After proposing a model for thick MVC, authors have implemented the model with varied technologies and then tested for response time; that has been improved significantly.

Keywords: Web 2.0 application, Thick-client MVC, Traditional MVC, Web application implementation

Introduction

The client interface of the traditional web application even based on MVC [Krasner G.E. and Pope S.T.], is simply an interpretation of output by the web browser that server generates. The server is solely responsible to refresh the client's view with each new request. Such general steps that could be performed after a request is transferred to server may be;

1) modifies the state of data (on server), based on client input,
2) constructs an HTML representation of relevant set of business object, and then
3) refreshes the client's view by writing the HTML to the browser.

The above general steps can be performed with the components Model, View and Controller, if the web application developed is based on MVC. The roles of above components as described by [Hall and Larry Brown] are:

- **Model:** Describes the part of application where data resides and is being implemented in the server side of the web application.

- **View:** Defines the presentation part of the application and resides on the client's web browser (Server keeps the view generating logic).

**Department of Computer Science, Jamia Hamdard, New Delhi, India.*

Authors are grateful to the anonymous referees for comments and suggestions for improving the text and contents of the paper. Authors alone are responsible for any errors/mistakes still remaining in the paper.

- **Controller:** Controls the flow of whole web application. It resides on the server. Based on the event triggered by the client, it coordinates with Controller and View.

In the traditional MVC approach, the main role of a client is to be a view of the model. Following figure sketches the architecture of this implementation approach.

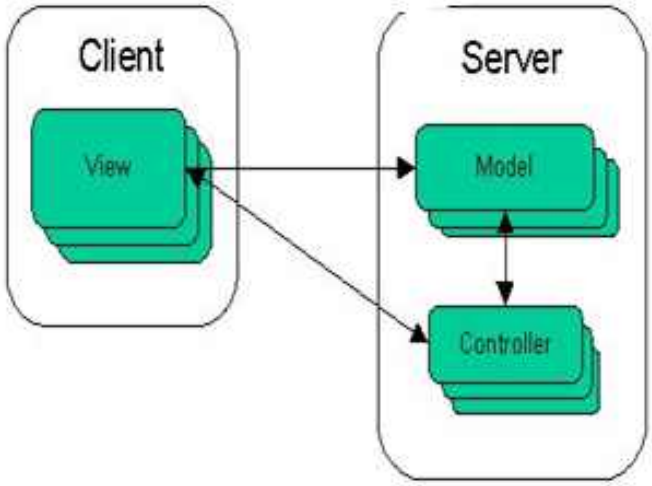

Figure 1: Traditional MVC

The availability of comprehensive client side technologies provides an opportunity to develop thick client MVC. The standardization of common MVC combining client side MVC and server side MVC is of great concern.

The present paper introduces an architecture termed Thick Client MVC that provides better performance in terms of response time than the traditional MVC approach.

Related Work

Many attempts have been made towards the said concerns. First of the notable work is published in [Katherine Betz] referred to as Dual MVC. The dual MVC was unnoticed and not got much acceptance by software fraternity due to the lack of technology support at that time. The authors claimed improvement of performance as the MVC on the client empowers client to generate the view in the certain cases. However, the development environment for such implementation has been left for future work. Another work on the same direction was published in [Avraham Leff and James T. Rayfield] as a FWAP (Flexible Web Application Partitioning). The FWAP proposed duplicate copy of Model; one for client and one for server. The entire Model loading at both client and server, leads to issue of duplicate overhead, maintenance and performance.

Further a research was published in [Michele Sama and Franco Raimondi] that proposed a synchronized symmetric MVC. The work proposed synchronized model. With the advent of web 2.0 and AJAX [Thomas A Powell] where the data transfer between the client and server is done asynchronously, the said work does not seem to be relevant for asynchronous communication between client and server.

One more work presented in [Janne Kuuskeri] focuses on the breaking of the web application into two. The full MVC had been implemented in client while server was completely decoupled. The server side implementation had been relied on only restful API. However the issue of server side MVC and its interaction with client side MVC has not been addressed in the said work.

Recently, a research was published in [Andreeva J., Dzhunov I., Kararvakis E., Kokoszkiewicz L., Nowotka M., Saiz P. and Tuckett D.] that shown the implementation of client side MVC with the event based communication between the components. The data source is being decoupled behind web API. [J andreeva, I Dzhunov, E Kararvakis, L Kokoszkiewicz, M Nowotka, P Saiz and D Tuckett] also described the client side components and their relationships to each other and with web API.

Proposed Model and Approach

Based on the suggestions of related works, architecture of Thick Client MVC has been proposed to exploit the potential performance improvement. MVC has been partitioned into two parts. Client implement a MVC on its own and is not fully dependent on MVC of the server side. The client not only comprises of View but also has its own Model and Controller.

The data mapping between the Model of client and server is being done asynchronously. Implementation of user interface may use the HTML. Controller can exploit AJAX technology that drives update in to Model of client, which is implemented by Document Object Model (DOM). The server side Controller may be implemented via servlets and the Model may be as JavaBeans.

The request from client to server is done asynchronously and the data in the form of XML/JSON is being responded to the client where Model and View are updated subsequently.

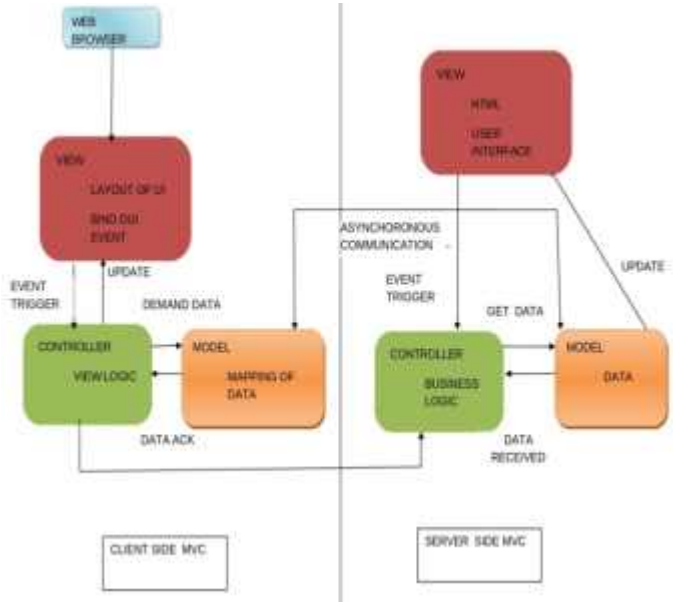

Figure 2: Proposed Model of Thick Client MVC

Implementation

To implement the proposed design MS-SQL has been chosen as a database, Tomcat 6.0 as publishing tools, Net Beans 6.7.1 as programming tools and JMeter [Emily H. Haili] as testing tool. Two web applications, one is based on proposed Thick Client MVC and another with traditional MVC, has been implemented.

The present work has taken a Customer look up as a case for the implementation. The customer look up is implemented in traditional MVC and Thick Client MVC. The response time of both the application is being recorded and comparative analysis is carried out.

a) Implementation of Controller in server side

During the development of J2EE-based system [Hall and Larry Brown], in order to take full advantage of MVC the required servlets has been designed and mentioned in deployment descriptor to act as a controller. The controllers of implementation are:

```
<!DOCTYPE web-app
PUBLIC  "-//Sun Microsystems, Inc.//DTD Web Application
2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
<servlet>
<servlet-name>AjaxZipCodesServlet</servlet-name>
  <servlet-class>
   controller.com.nit.ajax.servlet.AjaxZipCodesServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>AjaxZipCodesServlet</servlet-name>
  <url-pattern>/zipcodes</url-pattern>
</servlet-mapping>
<servlet>
  <servlet-name>AjaxSignupServlet</servlet-name>
  <servlet-class>
    controller.com.nit.ajax.servlet.AjaxSignupServlet
  </servlet-class>
  <load-on-startup>4</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>AjaxSignupServlet</servlet-name>
  <url-pattern>/signup</url-pattern>
</servlet-mapping>
<servlet>
  <servlet-name>AjaxLookupServlet</servlet-name>
  <servlet-class>
    controller.com.nit.ajax.servlet.AjaxLookupServlet
  </servlet-class>
  <load-on-startup>3</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>AjaxLookupServlet</servlet-name>
  <url-pattern>/lookup</url-pattern>
</servlet-mapping>
<servlet>
  <servlet-name>AjaxUsernameServlet</servlet-name>
  <servlet-class>
    controller.com.nit.ajax.servlet.AjaxUsernameServlet
  </servlet-class>
  <load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>AjaxUsernameServlet</servlet-name>
  <url-pattern>/username</url-pattern>
</servlet-mapping>
<welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

(i) Code Snippet of Controller in Server Side

b) Implementation of Model in server side

During the development of J2EE-based system, in order to take full advantage of MVC the required models as the plain java class has been designed to act as a model. The models of implementation are:

```
package model.com.nit.ajax.pojo;
import java.io.*;
import java.util.*;
public class UserProfile implements java.io.Serializable
{
  private String username;
   private String password;
  private String email;
  private String first_name;
   private String last_name;
   private String address;
  private String zipcode;
   private String city;
   private String state;
    private String type;
   public String getType() {
      return type;
   }
   public void setType(String type) {
      this.type = type;
   }
   public String getAddress() {
      return address;
   }
 public void setAddress(String address) {
      this.address = address;
   }
   public String getCity() {
      return city;
   }
   public void setCity(String city) {
      this.city = city;
   }
   public String getEmail() {
      return email;
   }
   public void setEmail(String email) {
      this.email = email;
   }
   public String getFirst_name() {
      return first_name;
   }
   public void setFirst_name(String first_name) {
```

```
    this.first_name = first_name;
  }
  public String getLast_name() {
    return last_name;
  }
  public void setLast_name(String last_name) {
    this.last_name = last_name;
  }
  public String getPassword() {
    return password;
  }

  public void setPassword(String password) {
    this.password = password;
  }
  public String getState() {
    return state;
  }
  public void setState(String state) {
    this.state = state;
  }
  public String getUsername() {
    return username;
  }
  public void setUsername(String username) {
    this.username = username;
  }
  public String getZipcode() {
    return zipcode;
  }
  public void setZipcode(String zipcode) {
    this.zipcode = zipcode;
  }
}
```

(ii) Code Snippet of Model in Server Side

c) Implementation of Views in server side

During the development of J2EE-based system, in order to take full advantage of MVC, the required views are java server pages that has been designed to act as a view. The views of implementation are:

index.jsp, manager.jsp, confirmation.jsp and failure.jsp.

d) Implementation of Controller in client side

During the development in order to implement controller in thick client MVC, system has been used with AJAX technology. The required controller has been implemented with the help of these technologies. The client side controllers of implementation are:

```
var zip = document.getElementById("zipcode");
                var url = "/ch05-suggest/zipcodes?zip=" +
escape(zip.value);
    name.value="?"+name.value;
    if (window.XMLHttpRequest){
        req = new XMLHttpRequest();
     }
     else if (window.ActiveXObject){
        req = new
     ActiveXObject("Microsoft.XMLHTTP");      }
    req.open("Get",url,true);
    req.onreadystatechange = callbackCityState;
    req.send(null);
Acting as a controller
```

(iii) Code Snippet of Controller in Client Side

e) Implementation of Model in client side

During the development in order to implement model in client side J2EE-based system has been used with AJAX technology. The required model has been implemented with the help of these technologies. As in AJAX technology, the response format from the server is XML/JSON [24]. In order to present the data in presentable format for client, a model in the client side is being designed to set/get the data. The client side model of implementation is:

```
 var jsonData = req.responseText;
debugInfo("jsonData"+jsonData);
var myJSONObject = eval('(' + jsonData + ')');
var password = document.getElementById('password');
password.value=myJSONObject.customer.PASSWORD;
var email = document.getElementById('email');
email.value=myJSONObject.customer.EMAIL;
var name = document.getElementById('name');
name.value=myJSONObject.customer.NAME;
var address = document.getElementById('address');
address.value=myJSONObject.customer.ADDRESS;
var zip = document.getElementById('zipcode');
zip.value=myJSONObject.customer.ZIPCODE;
var city = document.getElementById('city');
city.value=myJSONObject.customer.CITY;
var state = document.getElementById('state');
state.value=myJSONObject.customer.STATE;
```

(iv) Code Snippet of Model in Client Side

The JSON format is being used to pass the data to the client. At the client end, the data passes by the server side is being held in the model of the client side.

f) Implementation of Views in client side

The required views are java server pages that has been designed to act as a view. The views of implementation are:

index.jsp, manager.jsp, confirmation.jsp and failure.jsp.

g) Snap shot of implemented applications

The output of the customer lookup implemented in both the applications has been displayed below.

Figure 3: Snap Shot Of Customer Lookup In Thick Client MVC

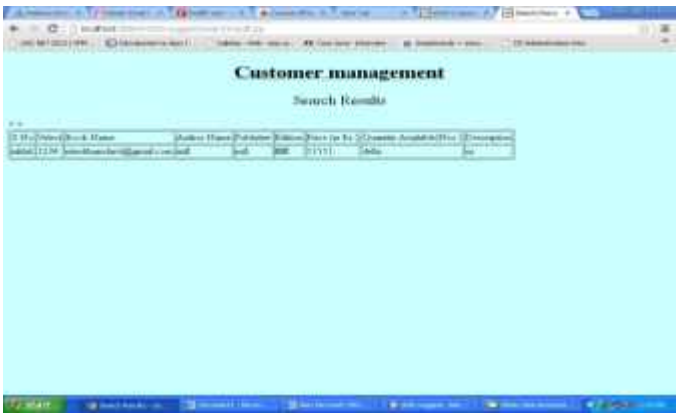

Figure 4: Snap Shot Of Customer Lookup in Traditional MVC

Results and Discussions

The proposed solution has been validated by data gathered by the testing tool after testing both the implemented applications. Tools used to validate the solution is JMeter. The data received by the testing tool for both the application for different users is displayed in the table and the line graph based on these data has been drawn.

Validation is based on the response time generated on traditional MVC and Thick client MVC by JMeter for different number of users and is analyzed. The MVC pattern of both the application is also analyzed on the code used in both the system respectively.

Validation data

Both the application has been tested by creating different number of simulated users by Jmeter testing tool. The average response time of each page of the application and the total response time of whole application has been captured by Jmeter. The total response time of both the application for simulated one user has been gathered from the Jmeter testing tool. The response time is being displayed in the subsequent snapshot

a) Response time of traditional MVC for one user

The average response time of J2EE application generated by Jmeter for one user is 1107 Millisecond. The average response time is obtained by calculating mean of all the response time of each pages of the application.

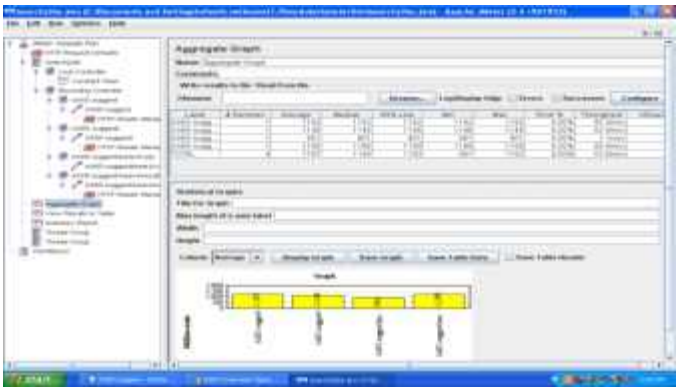

Figure 5: Jmeter Output of Response Time of Traditional MVC Application For One User

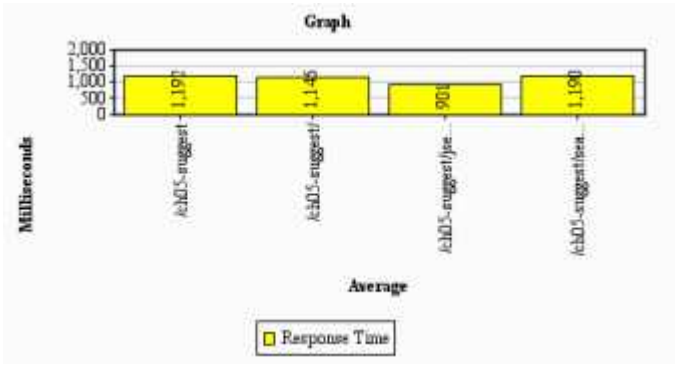

Figure 6: Graph of Response Time of Various Pages in Traditional MVC For One User

b) Response time of thick client MVC for one user

The average response time of J2EE with AJAX application generated by Jmeter for one user is 1080 Millisecond. The average response time is obtained by calculating mean of all the response time of each pages of the application.

Figure 7: Jmeter Output of Response Time of Thick Client MVC Application For One User

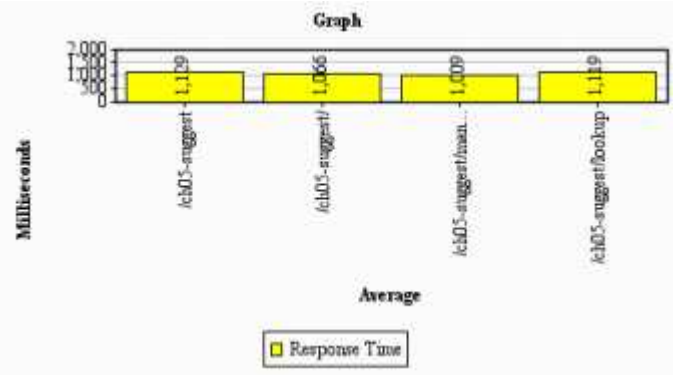

Figure 8: Graph of response time of various pages in thick client MVC application for one user

The data gathered from testing tool for different user has been placed in two dimensional tables and the line graph. Number of users as X-axis and Response in Millisecond as Y-axis has been drawn for the two implemented technologies.

RESPONSE TIME DATA		
No Of Users	J2EE response time (in millisecond)	J2EE with AJAX response time (in millisecond)
One user	1107	1080
Twenty-five user	2715	1267
Fifty user	2345	1268
Hundred user	2095	1255
Two hundred user	1993	1247

Figure 9: Table of Comparison of Response Time of Both The Application

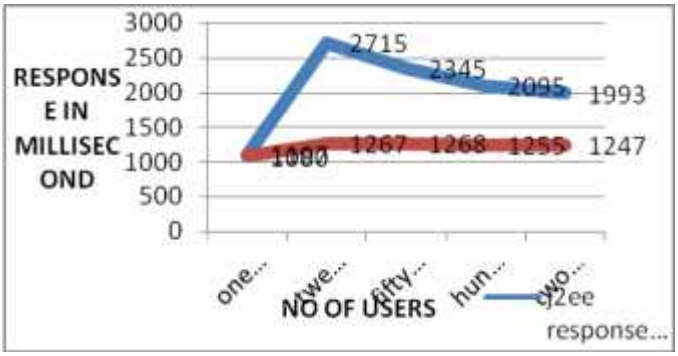

Figure 10. Graph of Comparison of Response Time of Both The Application

The graph showed that application build on Thick Client MVC has better and static response time than the application build on traditional MVC.

Conclusion

After the implementation testing of both applications with the help of JMeter showed that application build on thick client MVC has better and static response time than the application build on traditional MVC.

Web Application implementation can use the Thick Client MVC approach described in the present work to improve response time.

The present work can be extended with respect to through put, network transition and server performance with respect to load memory and CPU utilization. Development environment for implementation of proposed model can be explored.

Refrences

1. *Avaraham Left and James T. Rayfield (2001), "Web Application using MVC Design pattern," IBM research Report.*
2. *Time O'Reilly. (2007), "Define Web2.0 and understand its implications", Communication and Strategies no 65 page no 17.*
3. *William Crawford and Jonathan Kaplan. (2007), "J2EE Design Patterns" 2nd Edition.*
4. *Kristopher William Zyp (2008), "A AJAX performance Analysis", IBM research Report.*
5. *Osama Hiroshw Kazato, Rafael Weib, Shinpei Hagashi, Takashi Kabayasiand Motonshi saeki (2009), "Model-View Controller Architecture specific Mode Transformation".*
6. *Osama Hamed and Nedal Kafin (2009), "A "Performance Predication of web based Application Architectures Case Study .NET VS J2EE, "International Journal of Web Applications.*
7. *Ganapathi Nanjappa (2010), "Performance Analysis of Web Methods Integrations using Apache Jmeter", Torry Harris Business Solutions.*
8. *Zeliko Jovanovic and Ivan Miletic (2011), "J2EE MVC pattern Applications with efficient Ajax Support and Web Service Integration", Ministry Of Education and Science of The Republic of Serbia aided research.*
9. *Min Hu, Ding Ding Pan and Pei-En Zhou (2011), "Research and Application of J2EE and AJAX technologies in Industry report", Journals of Computers.*
10. *Andreeva J., Dzhunov I., Karavakis E., Kokoszkiewicz L., Nowotka M., Saiz P. and Tuckett D. (2012) "Designing and developing portable large-scale JavaScript web applications within the Experiment Dashboard Framework", International Conference on Computing in High Energy and Nuclear Physics.*
11. *Michele Sama and Franco Raimondi, "Snchronised Symmetric Model-View -Controller", http://www.rmnd.net/pubs/ssmvc.pdf.*
12. *Robbins", Scaling Isomorphic Javascript Code", http://blog.nodejitsu.com/scaling-isomorphic-javascript-code.*
13. *Avraham Leff and James T. Rayfield (2001), "Web-Application Development using the Model/View/Controller Design Pattern", Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing.*
14. *Peter Bell (2012), "A survey of Client MVC framework", IBM research paper.*
15. *Maorale-Chaparro et.al. (2007), "MVC Web Design Patterns and Rich Internet Applications", www.sistedes.es/sistedes/pdf/2007/SCHA-07-Morales-MVC.pdf.*
16. *Osama Hiroshw Kazato, Rafael Weib, Shinpei Hagashi and Takashi Kabayasiand Motonshi saeki (2009), "Model-View Controller Architecture specific Mode Transformation".*

17. *Katherine Betz (1999), "Developing Highly-Responsive User Interfaces with DHTML and Servlets", IBM research paper.*
18. *William Crawford and Jonathan Kaplan (2007), "J2EE Design Patterns", 2nd Edition.*
19. *Hall and Larry Brown, "Core Servlets and Java Server Pages", 2nd edition.*
20. *Janne Kuuskeri, "Experiences on a Design Approach for Interactive Web Application".*
21. *Emily H.Haili. (2008), "Apache JMeter", PACKT Publishing.*
22. *Krasner G.E. and Pope S.T. (1988), "A Cookbook for Using the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System", Journal of Object Oriented Programming, 1 (3): 26-49.*
23. *Thomas A Powell, "The Complete Reference AJAX",, Tata - Mcgraw Hill, http://ajaxref.com/*
24. *http://json.org/java/*