

---

## APPLYING HIERARCHICAL MVC ARCHITECTURE TO HIGH INTERACTIVE WEB APPLICATIONS

Micael Gallego-Carrillo, Iván García-Alcaide, Soto Montalvo-Herranz

**Abstract:** *This paper presents a very new architecture for developing high interactive web applications. At the present time, there are many applications based on web. To manage, extend and correct them can be difficult due to the navigational paradigm they are based on. From here we would like to contribute to make these tasks easier taking advantage of experience obtained from the development of standalone applications in the past. Therefore, we would like to begin to settle the concepts and tools for a new framework.*

*There are other frameworks and APIs offering MVC architectures to web applications, but we think that they are not applying exactly the same concepts. While they keep on basing their architectures on the navigational paradigm, we are offering a new point of view based on an innovator hierarchical model. First, we present in this paper the main ideas of our proposal. Next, we expose how to implement it using different Java technologies. Finally, we make a first approach to our hierarchical MVC model. We also compare shortly our proposal with the previously cited technologies.*

**Keywords:** *Web Applications Engineering, Model, View, Controller, MVC.*

---

### Introduction

When the web was created, its main goal was to provide to the scientific community the chance of sharing information through hypertext in a comfortable way. At the beginning, files supporting this information were completely static and it could only be changed modifying the content of the files manually. Nowadays, we find a great evolution at this respect and we can see really complex services offered by web sites. This way, we are able of managing complete applications based on web. This fact is more than interesting for giving the possibility of using an application almost everywhere through a network such Internet without installing complicated clients but a web browser.

This is the main reason for enterprises and other organizations to base their development interfaces in web, but this may not be so easy. Patterns and many other design tools look forward to find a way to make software products extensible, manageable, reusable and easy to support. MVC architecture appeared to reach this point for window based interfaces. Now it seems to be apprehensible to think if it would be possible to take MVC advantages to web based interfaces.

In this document we propose a MVC architecture for web applications based on Java 2 Enterprise Edition [J2EE, 2005]. First, we will describe the main concepts and organization of our proposed architecture. Although a brief glance to the used technology can be found at this point, this description would be applicable to any other development tool than Java. Next, we describe in detail the implementation of this architecture. Our main goal is to establish the basis to get a complete framework for developing high interactive web applications in a comfortable way.

---

### Description of the MVC Architecture in Web Applications

In this paper we present the base and main concepts to raise a framework to develop web interfaces for applications using MVC architecture [Krasner, 1988]. This architecture reduces the coupling between classes and increases their cohesion. This fact gets the code to be more independent so it can be easily reused to save time and effort in further developments. MVC has been widely implemented in graphics user interfaces to separate the entity responsible of showing information (view), the one responsible of storing it (model) and the one responsible of receiving user events (controller). The standard UI technology used in Java, Swing [Swing, 2005], also uses MVC.

---

As expected, the design we are writing about has three clearly separated parts, each one of them assumes a different responsibility corresponding to MVC model. They are the model, the view and the controller. As follows, we describe them:

### **The Model**

This component is the responsible for data maintenance. It has to keep it available for the application in a consistent and safe environment, not allowing any external or internal intervention to affect in any negative aspect. Every single code that implements the model has to be reusable by any other kind of interface without modifying it.

To implement this, we need a set of classes. In order to take advantage of many other technologies, these classes have to preserve JavaBeans specification [JavaBeans, 2005].

### **The View**

The view is the responsible for the application interface. In this case, it is the HTML and JavaScript code to send it to the client (or any other format like WML). It has to show and to receive the information managed by the application, so it can interact with the user. From here, it has to be given the chance of sending information to the application through events. It is considered a real event a request to the server, therefore, the view has to provide mechanisms like links and buttons to carry out the appropriate requests.

The way to implement this part is through JSP technology [JSP, 2005]. Every JavaBean of the model may have at least one JSP file to be presented. From these files it is used different technologies to access the data to be managed like expression language and tag libraries [Taglibs, 2005].

### **The Controller**

It is the responsible for controlling the interactions with the user. Whenever the application receives something produced by the user, the controller has to decide what to do next. This is the part that manages the global flow control of the application.

From a web browser, there is only one possibility of sending information to the server. This is done sending a request to a web resource. When requesting, the user can attach more information (for example from a form) than the reference to the new wanted resource. There should be one controller by each class that appears in the model and is presented through JSP files. The controller will be a Java common class with methods that receive data about what happened, interpret it and execute its code consequently to continue with the application.

### **The Events System**

In order to follow the MVC architecture, the user interacts with the application raising events. But the problem is that web pages have not the possibility of raising events more than requests. We call this kind of events real events. When an object of the model is presented it can give the chance of updating its information through a form. When the real event arrives to the controller, it carries the information from the form, so it can guess if something has happened or changed. We consider every single change between showed data to the client and received from it as an event and we reference them as deferred events pack. It is important to remark that the order or the time in which they are produced is not relevant at this point.

---

## **Implementing MVC with J2EE**

---

From the time that web was used for something than showing static information, developers had to solve a lot of problems derived from HTTP and HTML because they were not thought to support, so many things necessities to give dynamism to web. So, new concepts and tools were developed to make programming web applications easier and possible such sessions, cookies, etc. We will try to take advantage of these and many other technologies when considered appropriate to help the MVC implementation.

In this point it is explained how to implement our MVC model and what technologies we are using for it. In Figure 1 we show a scheme of this and in Figure 2 a web application request collaboration diagram.

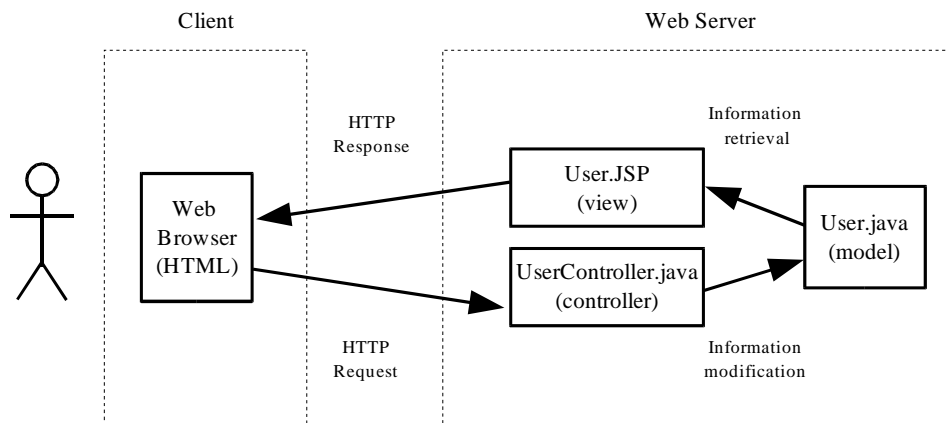


Figure 1. Scheme of the design. It shows the collaboration between different layers

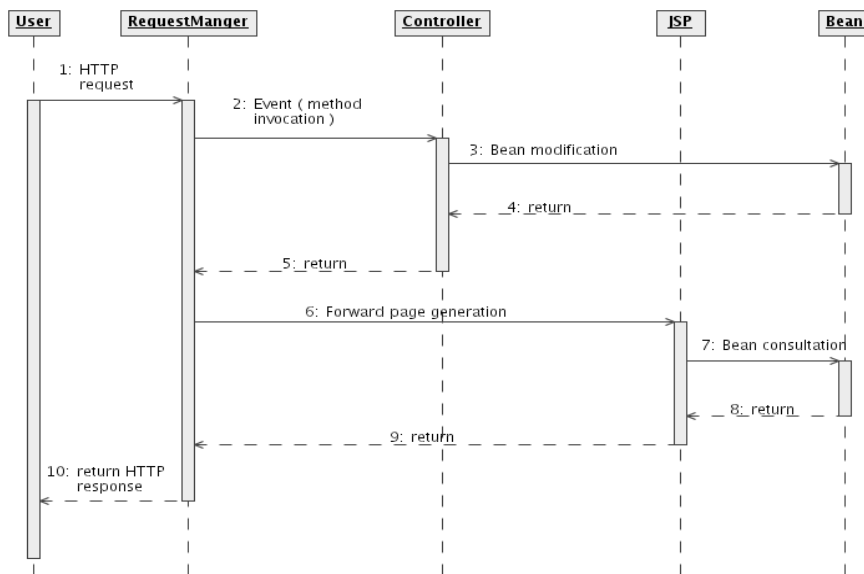


Figure 2. Web application request collaboration diagram

### Implementing the Model

In the model, we need Java classes that support the data to be managed by the application. Fulfilling JavaBeans specification will later help us to be able of using technologies that have JavaBeans as requirement. Information supported by these classes is contained in their attributes and the type of them can be another class of the same model giving rise a composition hierarchy or an association relation. Hence, basically these classes will have just some attributes and methods to access them.

In this component may appear classes related to persistent storage technologies, like data bases, if needed to get or store information in that way.

### Implementing the View

View implementation is mainly based on JSP files that access to model to show it. As classes in the model are JavaBeans, it will be more comfortable to apply Expression Language and User Defined Tag Libraries. One JSP file is always associated to a class in the model and its responsibility is to present the whole or a part of the information containing on it. One class of the model may be associated to one or more JSP files. It will be convenient to associate more than one JSP file, if it is needed, to represent the information with different formats or parts of the associated class.

Representing the application is not made from a navigational point of view what means that there is not a home page to navigate to the rest of application. Instead of, the representation of an object is sent to the user through HTTP. Because the represented object can receive events, it is its responsibility to give the appropriate links and forms to the client. If JSP file do this directly, it would be keeping the navigational paradigm, so links and form that generate real events should be generated dynamically. To get it, we let user-defined tags to take care of it. It has a code behind able of producing the text necessary for sending the request correctly to the server when the user click on a link or submit a form. Hence, we can not say that there are pages to navigate through, but there is the possibility of executing different code of the application according to the given request.

### Implementing the Controller

The controller will be a set of classes that receive information from the requests and execute the code of the application according to it. It is needed something to manage requests to the server in order to be executed them by the correct controller. This responsibility will be taken by the request manager, which is a Servlet that it is included in the framework. On the other hand, there will be a controller class by each JSP file that is able of sending requests. Because it is not necessary to keep the state of these classes, all their methods will be static. Every controller class has one method associated to each link or form that its JSP file is able to generate in order to attend it. Every method will receive one parameter with the object of the model that the JSP used to represent it. If there is additional information on the request like components of a form or parameters of a link, this information will be received by the method also in order to let the controller to make the oportune decisions.

### Hierarchical MVC Architecture

To build the component model using object oriented programming, it is often used composition hierarchies between classes. During the development of the user interface, it would be very useful to be able of using the same composition architecture. To get it, each class in the model has one or more JSP files in the view that will take the responsibility of presenting it. When the representation of one object has another object embedded by a composition relation, the JSP file associated to the first object will delegate to the corresponding JSP file associated to the related one its representation. With this approach, a JSP file may have to include another one or more if in the model one class has attributes of another model class type. It is showed in Figure 3. For implementing this approach, it is very useful to have another user-defined tag.

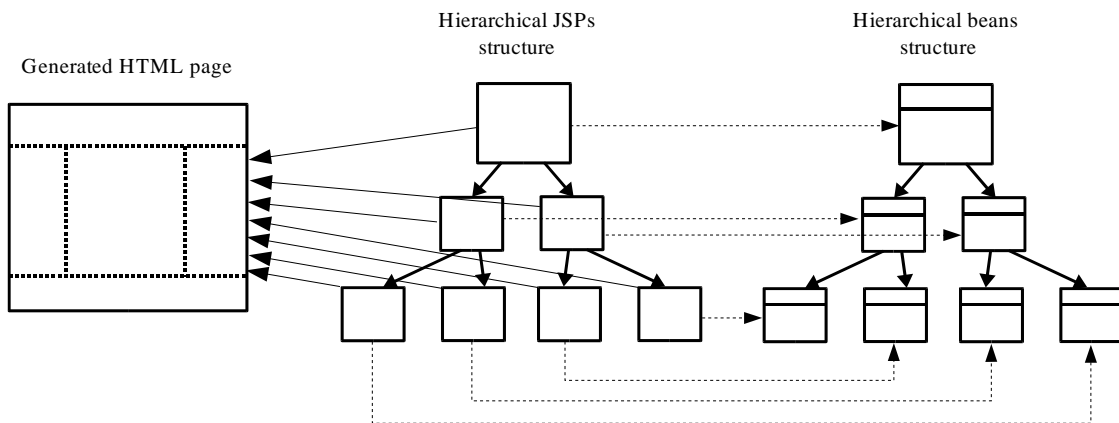


Figure 3. Hierarchical MVC architecture

### Related Works

Nowadays, there are several technologies that apply MVC architecture for developing web applications. Most representative in J2EE they are Struts [Struts, 2005] and Java Server Faces [JSF, 2005]. Both are very similar in the way they apply MVC architecture.

JSF applies it at page level; it is to say there is one JSP file per page. The model is represented by one object per page as well. The controller is implemented by the own technology and it is configured through defined tags in the

---

JSP file. The controller updates JavaBeans properties with the values from, found in the interface components generated by the JSP file.

Struts uses JavaBeans in the model also and it implements the view with JSP files that generate the output from those JavaBeans. The controller manages the requests received by the web application. It associates the actions to logic names used to build the links and forms in the JSP file.

Both technologies apply navigational paradigm and manage links through configuration files.

Main difference between these technologies and our proposal is that they have the navigation and page concepts as their base. In our approach, the page concept is not applied; instead of, we build a view from the information in the web application. Interactions with user make not him to navigate but to change the state of the application and when it is presented again, user can see the changes. Due to this approach, there is not one JSP file per page and when an object needs to be presented then, it will use its own JSP file. If the state of the object is based in other objects, it can delegate the presentation to them including other JSP files.

---

## Conclusion

---

In high interactive web applications, it is not possible to apply in a natural way the navigational paradigm because the user needs to see the state of the application on every moment. The architecture proposed here presents a set of classes that shows their content and modifications at a given time, so not navigational point of view is used in this approach. This fact helps a lot to increase the manageability of the project at developing time. On the other hand, this approach takes good care of encapsulation and reusability benefits offered by classes saving time in the develop process.

In the future, incorporating this architecture into an Integrated Development Environment will enable faster developing giving to the programmer the possibility of generating big parts of his code automatically. Moreover, building the user interface hierarchically, so it can be used the same delegation as in object oriented technology, makes that repercussions in changing code of the application are very limited.

Next to this work, we will continue developing tools, code generators and libraries while going deeper into these concepts to form a complete framework that will help in development of web applications with a high interaction.

---

## Acknowledgement

---

We would like to thank the following people for their help and their support: Luis Fernández Muñoz, Fernando Arroyo Montoro, José Ernesto Jiménez Merino and Carmen Luengo Velasco.

---

## Bibliography

---

[J2EE, 2005] Java 2 Enterprise Edition. <http://java.sun.com/j2ee/>

[Krasner, 1988] Krasner, G. and Pope, S., 1988. A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 system. *Journal of Object Oriented Programming*, Vol. 1, No. 3, pp 26-49.

[Swing, 2005] Swing. <http://java.sun.com/products/jfc/>

[JavaBeans, 2005] JavaBeans. <http://java.sun.com/products/javabeans/>

[JSP, 2005] JavaServer Pages. <http://java.sun.com/j2ee/jsp/>

[Taglibs, 2005] Tag libraries. <http://java.sun.com/products/jsp/taglibraries/>

[Struts, 2005] Struts. <http://jakarta.apache.org/struts/>

[JSF, 2005] JavaServer Faces. <http://java.sun.com/j2ee/javaxserverfaces/>

---

## Authors' Information

---

**Micael Gallego Carrillo** – ESCET, Universidad Rey Juan Carlos, C/Tulipán s/n, 28933 – Móstoles (Madrid), Spain; e-mail: [mgallego@escet.urjc.es](mailto:mgallego@escet.urjc.es)

**Iván García Alcaide** – LPSI, Universidad Politécnica de Madrid; Campus Sur, Carretera de Valencia Km. 7, 28031 Madrid, Spain; e-mail: [igarcia@eui.upm.es](mailto:igarcia@eui.upm.es)

---

**Soto Montalvo Herranz** – ESCET, Universidad Rey Juan Carlos, C/Tulipán s/n, 28933 – Móstoles (Madrid), Spain; e-mail: [soto.montalvo@urjc.es](mailto:soto.montalvo@urjc.es)