MVC ARCHITECTURE IN WEB APPLICATIONS DEVELOPMENT

Dominic Mircea KRISTÁLY
Adrian Virgil CRACIUN
Adrian PELCZ
Ion TRUICAN

Automatics Department, "Transilvania" University of Brasov, M.Viteazu Street, no.5, 500174, Brasov, Romania, phone/fax: +40 0268 418836,
kdominic@vision-systems.ro, craciun@vega.unitbv.ro,
apelcz@vision-systems.ro, itruican@yahoo.com

This paper presents an architectural model to use in order to develop easy-to-maintain multi-user web applications.

Nowadays, all industrial systems can be connected through a LAN to the Internet (The Informational highway), so it can be accessed through an interface by its users from any corner of the world. Modern web applications are built on multi-tier architectures, so the tasks are distributed between specialized modules.

For developing easy-to-maintain applications is recommended to use proven design patterns, such as MVC architecture.

The Java language offers three technologies that mold on the MVC architecture: Servlets, Java Server Pages and Java Beans.

Keywords: web, Java, JSP, MVC, communication

1. INTRODUCTION

Web applications tend to spread into every imaginable field of work, so a standard that assures easy ways to manage the code and customize the views of the users is always welcomed. Web oriented applications must allow simultaneous access for multiple users, each with different rights on the server's or LAN's resources.

A multi-user application must offer several views of the resources, according to the access rights attached to each user. This implies that every modification regarding the views or the business logic has a considerable impact on the application's source code.

Web applications are frequently built on a multi-tier architecture (commonly a three-tier one, with the levels: client, server and database). The tasks distribution in three-tier architecture is as follows:

- client side: the views;
- server side: the business logic;
- database: stores the data about the users and their rights or/and a list of resources with information about access rights.

Keeping these in mind, we need to find a smart way to separate the business logic from the views (the interface that presents the resources to the user) so the updating of source code can be done easily and fast without the need of modifying all the application's modules.

When developing applications, web-oriented or not, it is better to use proven design patterns, tested by many developers. One of these *design patterns* is the MVC architecture, described by *Smalltalk* in the 70's. Since that time, the MVC idiom has become commonplace, especially in object-oriented systems.

The MVC architecture, for the long version *Model-View-Controller*, unites a set of rules to use when developing computer programs, in order to create easy-to-maintain applications.

The main idea is to separate the business logic of the application from its views and its data access modules. In this way, any changes made to the code (changes to the algorithms or sequence of processing operations), data access methods or views, will not determine modifications in others components of the application. Let's say is like making an abstract interface between the application's modules.

The diagram shown below presents the links between the *Model* (the business logic), the *Controller* (handles the events) and the *View* (presents the data provided by the Model to the client, in an ordered fashion, as a result of a request / event). In this context, the *Display* represents the client.

The goal of the MVC design pattern is to separate the application object (model) from the way it is represented to the user (view) from the way in which the user controls it (controller).

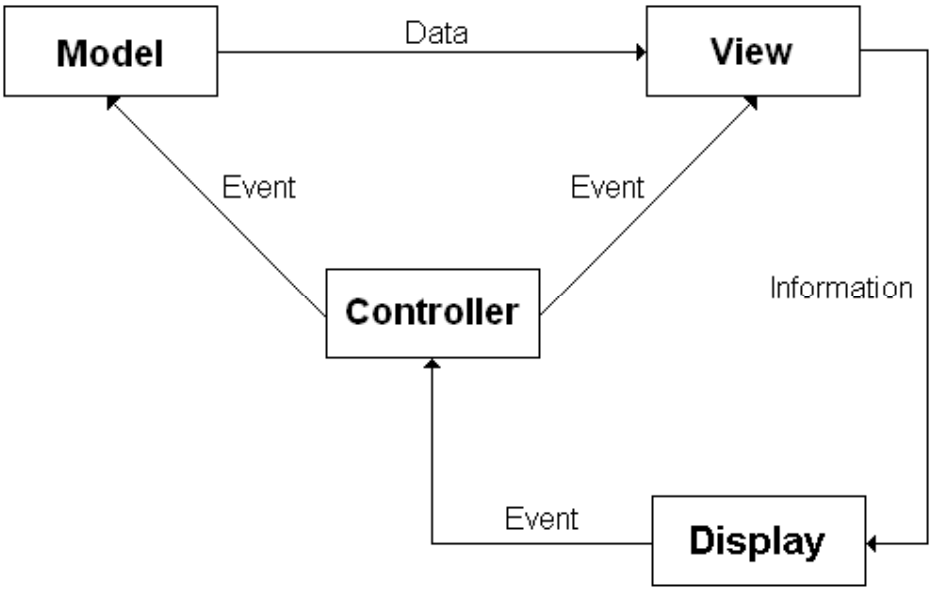

Fig. 1. Generic MVC Architecture

The goal of the MVC architecture is to separate the application object (model) from the way it is represented to the user (view) and from the way in which the user controls it (controller).

2. COMPONENTS DESCRIPTION

2.1. The Model

The Model includes the business logic of the application. In this way, is easy to test and debug the application.

The Model doesn't have to deal with the *outside world*: the way the data is presented to the user or how can he interact with it. The Model receives inputs and computes outputs. The types of the inputs and outputs depend only of the programming language used in the development of the application. The reusability of the code generated in this way is maxim.

The code written for this component solves the problem for which the application was created.

The data are accessed and manipulated through methods that are independent of the GUI (Graphical User Interface).

This part of the application is most unlikely to change in the application's lifetime.

The model could be split in other two subcomponents:

- the declaration section: defines the variables and the values used by the model and implements methods for modifying them;
- the action section: defines the accepted changes of the variables when an event occurs.

2.2. The View

The View represents the presentation part of the application: the way in which the result data provided by the Model is presented to the user. The View is the interface of the Model.

The View can be represented by a GUI, a CLI (Command Line Interface), or other means of communication between man and machine.

In multi-user applications, each type of user has to have a view associated with it, based on the access rights to the resources.

By separating the View from the Model it is possible to create user interfaces with different look-n-feel, without the need of changing the way the data is processed. All the interfaces will interact with the same model.

The view uses the query methods of the model to obtain data from it and then displays the information. A view renders the contents of a model.

2.3. The Controller

The Controller creates the unity of the MVC architecture. The Controller receives the events, determines the flow of actions proper to each situation.

The role of the Controller is that of a dispatch center for the events. The tasks it must fulfill are:

- Security: it is responsible with the authentication of users;

- Event identification: it must identify the arose event;
- Preparation of the Model;
- Event processing: it launches, according to an event map, the proper procedure attached to the arose event;
- Error handling;
- Dispatch of response.

3. MVC ARCHITECTURE USING JAVA TECHNOLOGIES

Java language offers three technologies for implementing web applications using the MVC paradigm: Java Servlets (extension modules for web servers like *Apache Tomcat*), Java Server Pages (JSP – server side scripting language) and JavaBeans (Java classes with a standard structure).

Each technology molds on certain components, as shown on the diagram below.

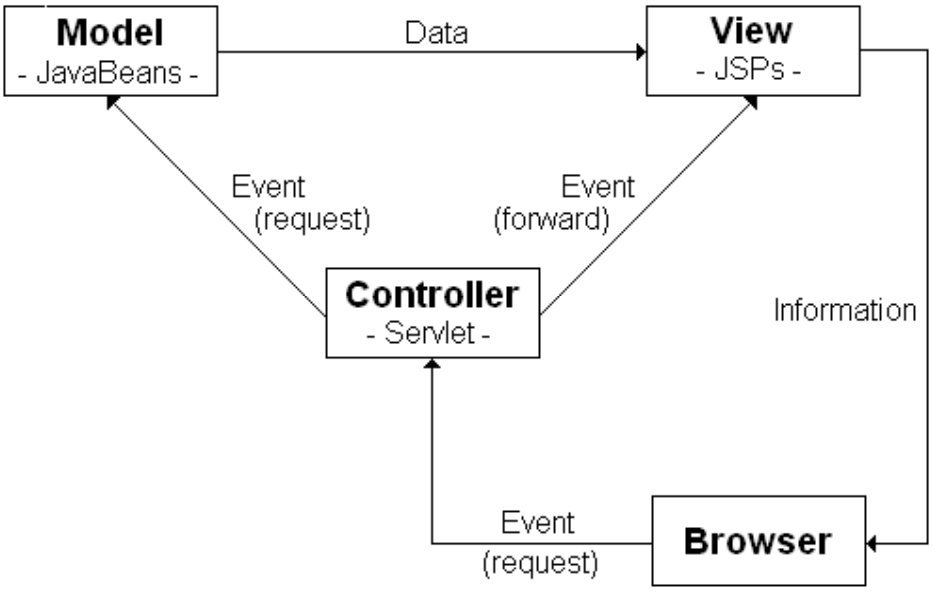

Fig. 2. MVC Architecture using Java technologies

Java Beans are used for the declarative section of the Model. The action section can be implemented using Servlets.

JSP pages are perfect for implementing the view. The generated pages are displayed by the client browser.

The controller can be easily implemented using Servlets.

These three technologies inherit all the advantages of Java language: simple, object-oriented, platform independent, stable, distributed, multithreaded.

4. CONCLUSIONS

The MVC architecture has the following benefits:

- **Multiple views using the same model:** the separation of model and view allows multiple views to use the same model. Consequently, an application's model components are easier to implement, test, and maintain, since all access to the model goes through these components;
- **Easier support for new classes of clients:** to support a new class of clients, all that is needed is to develop a view and the controller for it and wire them into the existing model;
- **Clarity of design:** the model's public method list make it easy to understand how to control the model's behavior;
- **Efficient modularity:** the components can be added / removed very easily, without negative influence on the application. Changes made to one component of the application aren't coupled to other components, eliminating debugging situations. The development of the various components can progress in parallel, once the interface between the components is clearly defined;
- **Ease of growth:** the application may incorporate an undefined number of modules, along side with old components, without interference;
- **Distributable.**

5. REFERENCES

[1] Brown, S., , *Professional JSP 2nd Edition*, Ed. Wrox Press Ltd., 2001
[2] Bergsten, H., *Java Server Pages*, Ed. O’Reilly, 2001
[3] Englander, R., *Developing Java Beans*, Ed. O’Reilly, 1997
[4] Flanagan D., *Java in a Nutshell,* O’Reilly, 1998
[5] Horstmann C., Cornell G., *Core Java Fundamentals (vol I and II)*, Sun Microsystems Press, 2001
[6] Sun Microsystems, *Java 1.4.2. Electronic documentation*