

MVC Web design patterns and Rich Internet Applications

Morales-Chaparro, R.; Linaje, M.; Preciado, J. C.; Sánchez-Figueroa F.
QUERCUS Software Engineering Group
Escuela Politécnica. Universidad de Extremadura
10071 Cáceres
{robermorales; mlinaje; jcpreciado; fernando}@unex.es

Abstract

Looking for the best design pattern to develop an application is not a trivial task and it depends on the requirements, target platform and experience of the development group among others. However, general guidelines for the design patterns issue have aided us during the last years. In this sense, we recommend a general purpose MVC design pattern for Rich Internet Applications that will allow taking advantage of all the current features of these applications, facilitating engineering processes.

1. Introduction

Nowadays, the growth of the Internet is a reality and it has great impact on business, industry, commerce and society [4]. The Web has become a common platform for providing access to information, therefore many of the old desktop applications tend to be developed as Web applications.

The Web Engineering community has proposed methodologies and tools to support the design, development, and maintenance of these Web applications [7] and they can be also specified at a high level of abstraction to facilitate code generation [14].

However, traditional Web applications are inadequate to respond to the new functionalities that users demand within them [17]. Web-tool vendors and Web developers are trying to answer such needs through the introduction of Rich Internet Applications (RIAs) which increase client-side storage and process capacities [2].

This new kind of applications is so novel that currently there is a lack of full development methodologies and architectures in this area [17]. From this situation, problems of maintenance and reusability may easily arise (as happened previously with desktop and Web applications)

which, at worse, could end in a new software crisis [4].

In this paper we discuss a general purpose design pattern, based on Model-View-Controller (MVC), which is a standard for interactive applications [18] and a generalized solution to recurring design problems in software development. This paper is mainly motivated by the lack of contrasted literature on Rich Internet Applications and design patterns at present.

To show the validity of the proposal, we present a multi-step Web application case study used throughout the paper, over different client-side rendering technologies in order to present advantages and disadvantages of each MVC design pattern. The proposed design could be applied to large-scale RIAs and developers could decrease time and costs of engineering processes.

This document is structured as follows: the main features of RIAs are presented in Section 2. Next, section 3 presents a brief introduction to MVC. In section 4 we show three different Web applications MVC approaches and we conclude it fixing a RIA MVC that is able to take advantage of the current RIA features. Finally, in Section 5 conclusions are drawn.

2. RIA overview

Traditional Web applications (based on HTML) have been extended in several directions (improving multimedia contents, interactivity, data retrieval...).

In particular, Web Interactive Applications (WIAs) [3] are defined as Web applications where a client program uses the Web infrastructure in order to reach end-users through Web based GUI (that usually requires a plug-in installation on the client-browser). RIAs fit well in this category but they present new features that have not been present together inside Web applications before, since RIA includes functionality inherited from

multimedia desktop applications such as drag-and-drop among others.

RIA does not try to replace HTML, because this language still remains perfect for representing simple data (including graphs and photos) without high interaction levels.

The concept of richness in RIA extends the traditional Web in three main aspects: data, presentation and communication capacities. These traditional Web applications improvements involve both, client and server.

Richness of data means users may manipulate a more sophisticated data model at client-side, reducing network communications, improving data refreshment or data filtering and the effective integration of multimedia contents among others.

Richness of presentation model provides advantages on user interactions, facilitating the User Interface (UI) manipulation avoiding full UI refreshments.

Richness of communication allows synchronous or asynchronous connections, message-based communications, etc. New possibilities of connections, as asynchronous data retrievals or synchronise clients in collaborative Web applications are native in RIAs.

RIAs can be implemented over multiple development platforms such as Adobe Flex or OpenLaszlo [2]. However a set of RIA features is commonly accepted by developers [2] [5].

3. MVC overview

The MVC design pattern was introduced in Smalltalk-80 and widely used in software design [6]. MVC improves scalability and maintenance, easing personalization, etc. [8].

The main benefits we expect from using design patterns include guaranteeing the quality and decreasing time and cost of engineering processes [13].

Dealing with interactive applications, Model-View-Controller (MVC) is a standard for design and a generalized solution in software developments [18] and RIA, as an interactive application is not an exception as our previous developments [16] have shown.

MVC separates the domain model (Model), the presentation model (View), and behaviours (Controller). General well-known MVC schema is depicted in Figure 1.

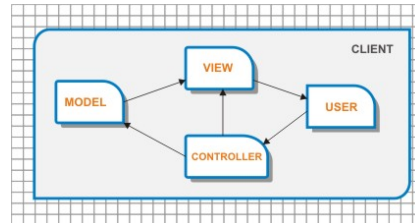


Figure 1. Desktop MVC

In general desktop applications, *Model*, *View* and *Controller* are usually programmed under the same language (e.g. C++) and they compose one executable file plus libraries (usually OS dependent to take advantages of UI or input functions). *Model*, *View* and *Controller* are at client-side, so the application and the User are at the same side [19].

Some specific frameworks are available to respond to RIA necessities, but the contribution of this paper is different, showing a general purpose MVC architecture that is not focused on a particular issue.

4. MVC and Web applications

The evolution of Web applications development is driven by the desire to modularize crosscutting concerns and decrease the dependency between Web designers and Web programmers [9].

A traditional Web application is generally an HTTP gateway to certain server-side resources (e.g. XML files, databases, etc.) where request parameters are processed by the server and passed to the Web application, which generates an HTTP response.

In this paper we will focus on large-scale Web applications which are data dynamic and typically written in at least two programming languages. The presentation is described in a client-side language (e.g. XHTML) and the functionality is specified using a server-side language (CFML, JSP, etc.).

In traditional Web applications, the domain model is stored at server(s) and the user makes low-level interactions with software running on the client, usually through a browser that renders the HTTP response at client-side. So in traditional Web applications development, *Model*, *View* and

Controller are distributed across the server/s and the client/s over the network [12].

We will show a case study used throughout the paper on the different MVC alternatives. In each of the following sub-sections are depicted at least two figures to show the MVC diagram and the implications of this MVC design pattern over the selected case study.

The selected case study is a typical four steps hotel booking application, because it allows advantages to be taken of some RIA goals like simplifying selections, multiple configurations and different purchase options. Also, it is simple enough to demonstrate advantages and disadvantages of different MVC design patterns. Booking steps are organized as follows:

- Step 1: date selection and number of rooms
- Step 2: selection of room type
- Step 3: summary and payment information
- Step 4: confirmation page.

4.1. Server-side MVC

This alternative is based entirely on server-side (depicted in Figure 2) and it is derived directly from desktop MVC, providing the server with all the computations and operations. A clear example of this situation is given in [9] [20] among others.

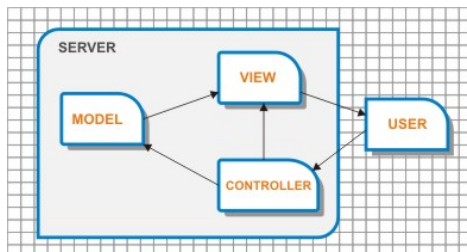


Figure 2. Server-side MVC

In this MVC design we must note that when we talk about User at client-side, we talk about a browser that provides an interface visualization and hypertext navigation. This MVC design pattern creates two cycles in the graph:

1) *User* → *Controller* → *View* → *User*

The *User* interacts with the application and this is sent to the *Controller* (to the server as a request) that responds sending a message to the *View* that updates its information and sends it to the *User* (to the client as a response).

2) *User* → *Controller* → *Model* → *View* → *User*

In this cycle, *User* interacts with the application and this is sent to the *Controller* that updates the state of the *Model*. Finally the *View* is updated to match the *Model* and sends it to the *User*.

This MVC choice is very useful when application capabilities requirements are poor at client-side (e.g. mobile browsers without JavaScript functionality) or when application necessity is only to display information with low levels of user interaction [21]. But we must be careful using this design pattern for all kinds of Web applications because:

- It can not update part of a *View*, it needs to create a new presentation at server-side and sends it to the *User*.
- It increases bandwidth usage in Web applications depending on interaction necessities.
- It causes the server to be busy all the time, not only serving pages, but also making form validations, generating UIs, etc.

The server-side must check two kinds of information retrieved from client-side: that all the form input fields are valid (e.g. a valid email) and that user selections are available (e.g. availability of rooms for selected dates).

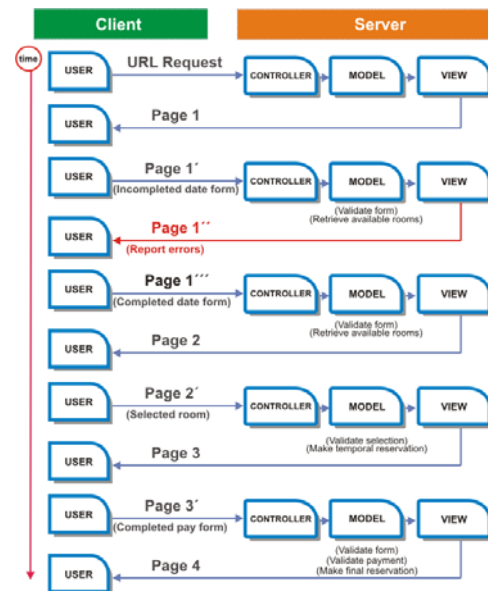


Figure 3. Network communication in server-side MVC

Communication between server and clients will increase dramatically if the number of users is too high. Figure 3 shows the communication between one client and one server with only one error (form validation) in page 1. Figure 3 represents a bad user experience: waiting for refreshments, obtaining unavailability over their selections... but even a bad company experience, because they lost clients through booking problems [5] and money because they pay per bandwidth usage.

One advantage in the Server-side MVC is that there is only one *Model*, one *View* and one *Controller*, so it is not too difficult to maintain the integrity of the *Model*, at least no more than in desktop applications. Also, all the application may be implemented under one server-page language generating HTML for clients.

4.2. Mixed client-side and server-side MVC

With this mixed approach, designers tried to focus their attention solving the problems that we have shown in section 4.1 derived from user interaction and server overriding.

In this way designers chose to use the client-side processor to do some validations on the data, usually using a client-side language supported by the browser like JavaScript or VBScript.

It's quite easy to make simple validations with script languages that run at client-side, but the main problem derives from the browser compatibility. Access to the UI elements is different among diverse browsers (i.e. Mozilla and Internet Explorer) and sometimes it depends on the clients' operative system. Browser vendors do not implement exactly the same standard specifications and they have their own bugs [15].

This situation is solved using client-side system detection and browser-dependent validation functions (i.e. one snippet of code for IExplorer, other for Mozilla, etc.). This solution increases the "page weight" that is sent to the client wasting network resources. Also there is a lack of full script documentation support for each current browser version available and this situation is worse when we talk about supporting older browser versions.

Forgiving these technological problems, a good development choice that has been largely applied is to use the MVC design pattern that

shows Figure 4. This design divides *Controller* and *View* between the server and the client-side [1] leaving the *Model* at server-side.

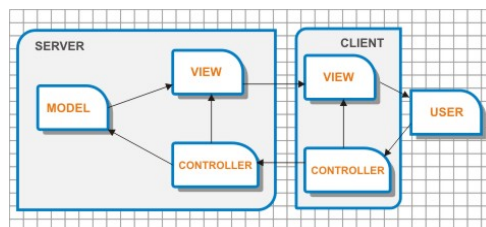


Figure 4. Mixed client-side and server-side MVC (1)

The mixed MVC design is more complex than the server-side one, due to the fact that it gives more functionality to the client. This MVC design pattern creates three cycles in the graph, one more than in the server-side MVC:

1) $User \rightarrow Controller (Client) \rightarrow View (Client) \rightarrow User$

The *User* interacts with the application only at client-side, so the page is sent to the user only once and later on, the user can interact with the interface obtaining quick responses since he does not need to access the server-side for these operations. The *Controller*, at client-side, must be capable of taking decisions about the scope of the required operations triggered by user interaction. If these operations require changes on the *Model*, the application control will give the control to the server *Controller*, but if they affect only the UI, *View* is updated and sent to the *User* at client-side.

2) $User \rightarrow Controller (Client) \rightarrow Controller (Server) \rightarrow View (Server) \rightarrow View (Client) \rightarrow User$

The *User* interacts with the application and this action fires the *Controller* at client-side that decides to send it to the *Controller* at server-side. Finally, the *View* at server-side is updated and sent to the client-side *View*. This is not a very usual situation, because *View* changes at server-side are usually ruled by changes in the *Model* since presentation changes by client interaction are possible, but hard to code.

3) $User \rightarrow Controller (Client) \rightarrow Controller (Server) \rightarrow Model \rightarrow View (Server) \rightarrow View (Client) \rightarrow User$

The *User* interacts with the application and this is sent to the *Controller* at client-side that fires the *Controller* at server-side which updates the state of the *Model*. Finally, the *View* is updated

to match the *Model* at server-side and the new UI is sent to the *User*.

This mixed design pattern is produced by several current frameworks and it generates a special type of HTML known as DHTML (Dynamic HTML = HTML + JavaScript) [1].

The mixed MVC design approach decreases bandwidth usage and the server overriding that arise in the server-side MVC (Figure 5), because form validations and conditions are processed at client-side.

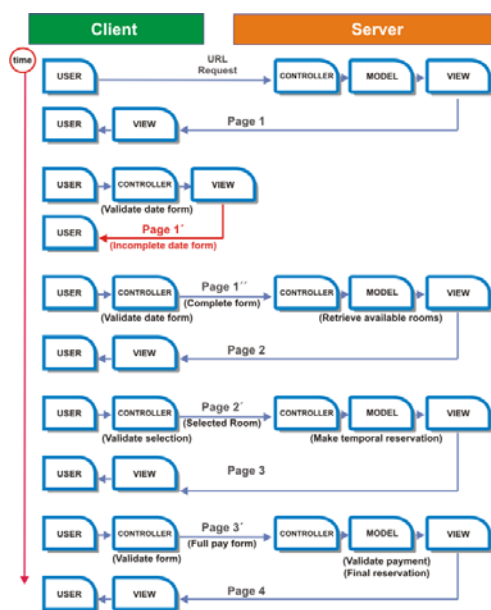


Figure 5. Network communication in mixed MVC (1)

But Web applications that use this mixed MVC still have several limitations, such as:

- limitations on making asynchronous data retrievals with the *Model* at server-side without page refreshment.
- limitations on providing high interaction levels, since there is no *Model* at client-side.
- difficult data manipulation, because without at least part of the *Model* at client-side, business logic or data is not available at this side.

And new problems arise:

- additional communications have appeared from request/response HTTP between *Views* and *Controllers* at client and server sides.
- the cycles that cross the client-server architecture (the last two that we have

explained) are more expensive in computation time and complexity than in the server-side MVC approach since there is piece of business logic at client-side.

We can solve the first problem of this mixed MVC design pattern leaving part of the *Model* at client-side. This solution usually involves migrating snippets of code from the *Controller* at server-side to the client-side (Figure 6) and it requires a browser with higher capabilities.

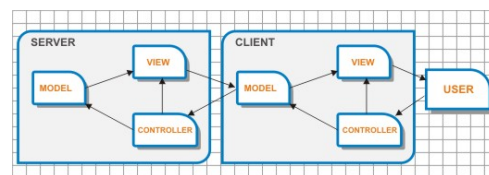


Figure 6. Mixed client-side and server-side MVC (2)

Again this design is not problem free. The main problems of this second mixed MVC approach are:

- computations for tasks that require access to the MVC at server-side are slow, since they need to commute through all the client MVC elements.
- the *Model* at client-side is extremely hard to program and maintain because, as we have explained before, it is necessary to deploy different codes per client platform (browser, operative system and device), with the associated problems of development maintenance and testing among others.
- Also one more complex cycle appears, so we can be sure that over-heading on communications among MVC design elements will appear.

This last mixed MVC is widely used by those current frameworks that have been adapted from sever-side design pattern to support advanced user interactions at client-side. This design pattern is able to be used for AJAX (HTML + JavaScript + Asynchronous XML data retrieval) and other similar technologies [10]. Currently our group is collaborating with one software factory and recent projects have been developed using AJAX and this MVC design pattern.

With this last design approach, if the user is able to use a RIA plug-in at client-side, increasing even more client-side capabilities, developers may decrease code development efforts and eliminate multi-version problems [17]. Anyway, with RIA

as client rendering technology, *Model*, *View* and *Controller* are divided between the client and the server-side and problems like additional commutations continues unsolved.

4.3. RIA MVC

To reduce problems derived from the server-side MVC and the two mixed MVC design patterns for RIAs, the alternative seems to give more functionality to the client-side, so the whole *Controller* and *View* of the application will be placed at client-side and also an important piece of the business logic (as part of the *Model*).

Figure 7 shows the RIA MVC approach that solves some problems we have presented before, but it remains the problem of multiple client-side platforms. Also it requires having different versions of the same application adapted for each client-side platform, but as we have mentioned at the bottom of section 4.2, plug-ins may solve this situation extending browsers capabilities.

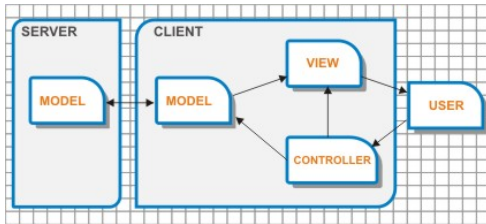


Figure 7. RIA MVC

The RIA MVC design pattern that we propose creates three cycles in the graph:

1) $User \rightarrow Controller \rightarrow View \rightarrow User$

The *User* interacts with the application only at client-side, so the page is sent once and the *User* can interact with the interface with quick responses since he does not need to access to the server-side. The *Controller* is able to indicate changes in the *Model* at client-side, so high levels of interaction with the showed data are available too.

2) $User \rightarrow Controller \rightarrow Model (Client) \rightarrow View \rightarrow User$

The *User* interacts with the application and this is send to the *Controller* that updates the state of the *Model* at client. Finally the *View* it updated to match the *Model* and sent to the *User*. All these operations are only at client-side, so network

bandwidth usage is minimized (in these two first cycles).

3) $User \rightarrow Controller \rightarrow Model (Client) \rightarrow Model (Server) \rightarrow View \rightarrow User$

Different interpretations of this cycle are possible. It can be seen as a complete cycle or not, because the communication between the *Models* at client and server sides is able to be asynchronous in RIA (transparent to the user). In this case the *Model* at client-side is the one that takes the decision to access the server-side *Model* e.g. if new data is required or to maintain the data of the UI updated.

This MVC design decreases network communication and server computation (Figure 8), taking advantage of the processing capabilities and resources (processor, memory...) available at client-side.

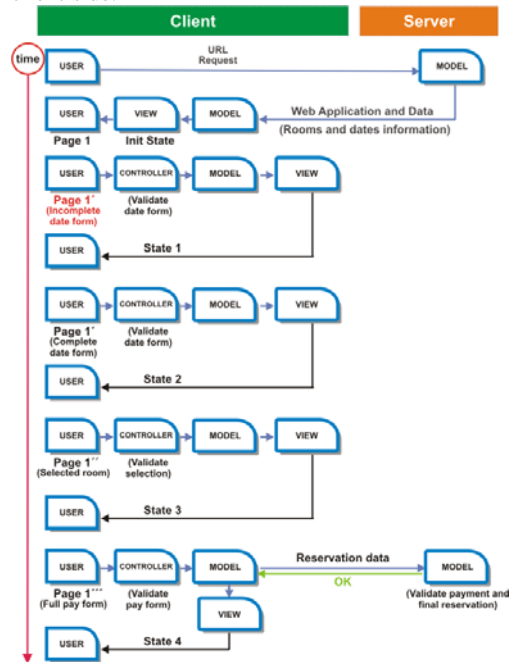


Figure 8. Network communication in RIA MVC

Problems derived from the server-side MVC design approach are solved with this approach:

- user can update part of the *View* at client-side (avoiding display refreshments).
- there is a decrease of bandwidth usage, as shows a simple comparison between figures 6 and 9.

- the server is ready to serve the application to new users nearly all the time while current users interact and receive feedback only from the client-side application.

Also, problems that arise from the mixed MVC approaches are successfully solved with this RIA MVC design, which has capabilities to:

- make asynchronous data retrievals with the *Model* at server-side.
- provide high interaction levels, not so limited as before, since there is an important piece of the *Model* at client-side.

Additional problems that arise from the client-server mixed MVC design, like additional communications and context commutations, do not continue within this approach.

Some RIA developers tend to leave all the *Model* at client-side, and this is possible (having the full *Model*, *View* and *Controller* at client-side). However, from our experience it is better to distribute the *Model* between the client and the server-side, because security and privacy issues in RIA are not completely solved at the moment.

As readers may observe, part of the client-side design pattern is included in the second mixed design pattern (compare figures 7 and 8), so some of the technologies mentioned in section 4.2 (e.g. AJAX) may use this design pattern too.

Finally, in order to complete our MVC client-side design pattern proposal, we propose a deeper separation within. This proposal (Figure 9) extends our client-side proposal discussed above with many benefits that we will enumerate.

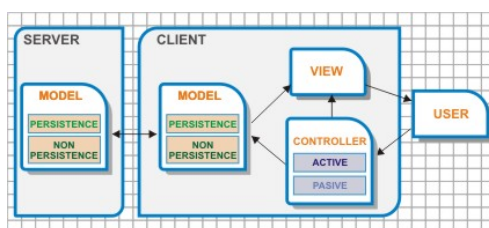


Figure 9. RIA deeper MVC

This MVC design divides the *Model* at client and server sides in two sub-layers, so:

- The server-side *Model* which involves two internal sub-layers: 1) Persistence layer that maintains non-volatile data, 2) Non-persistence layer that maintains volatile data.
- Client-side *Model* which involves two internal sub-layers: 1) Persistence layer: maintains

data retrieved from the server-side *Model*, 2) Non-persistence layer that maintains volatile data at client-side.

Also this design pattern divides the *Controller* into two sub-layers:

- Active *Controller* which takes control from user interaction (user triggered).
- Passive *Controller* which takes control from predefined actions (time triggered).

The main problem of this approach, used in some of our previous developments [16], is that it increases the developing time. However, there are other important advantages, mainly:

- maintenance time decreases
- quantity of reusable code increases
- the separation of tasks facilitates development efforts.

5. Conclusions

As usual, one conclusion about MVC is clear: MVC design pattern election for an application depends on the nature of the application and the analyst and developers experience. Moreover, not all kinds of applications have client processing or storage necessities.

We have discussed different general Web MVC design patterns throughout the document, which stand for different kinds of Web applications in order to adapt them to RIA.

RIA could use any of the design pattern proposals presented, however to take full advantage of the client-side capacities, we have proposed a general purpose client-side design pattern. This design optimizes commutations inside the MVC design and takes all the advantages from the previous ones towards RIA.

Finally, we propose a deeper separation within our RIA MVC proposal that improves some lifecycle stages of the application. Nowadays we use this proposal in many RIAs with very nice results.

6. Acknowledgements

This work has been carried out under the projects PDT06A042 y TIN2005-09405-C02-02.

Referencias

- [1] Betz K., Leff A. and Rayfield J.T., Developing highly-responsive user interfaces with DHTML and servlets, IEEE International Performance, Computing, and Communications Conference, 2000
- [2] Brent S.: XULRunner: A New Approach for Developing Rich Internet Applications. Journal on Internet Computing, IEEE, 2007
- [3] Christodoulou S. P., Styliaras G. D. and Papatheodorou T. S., Evaluation of Hypermedia Application Development and Management Systems, ACM conference on Hypertext and Hypermedia, 1998
- [4] Dart S., Configuration management: the missing link in Web engineering, ACM Books, 2000
- [5] Duhl J., Rich Internet Applications white paper", http://www.macromedia.com/platform/whitepapers/idc_impact_of_rias.pdf, 2003
- [6] Gamma E., Helm R., Johnson R. and Vlissides J., Design patterns: elements of reusable object-oriented software, Addison-Wesley Reading, 1995
- [7] Ginige A. and Murugesan S., Web engineering: an introduction, IEEE Multimedia, 2001
- [8] GuangChun L., Lu W. and Hanhong X., A novel Web Application frame developed by MVC, ACM SIGSOFT Software Engineering Notes, 2003
- [9] Kojarski S. and Lorenz D.H., Domain Driven Web Development With WebJinn, ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, 2003
- [10] Leff A. and Rayfield J.T., Web-application development using the Model/View/Controller design pattern, IEEE Int. Enterprise Distributed Object Computing Conference, 2001
- [11] Linaje, M., Preciado, J.C., Sánchez-Figueroa, F.: A Method for Model Based Design of Rich Internet Application Interactive User Interfaces, Int. Conference on Web Engineering, LNCS, 2007, to be published
- [12] Morse S.F. and Anderson C.L., "Introducing application design and software engineering principles in introductory CS courses: model-view-controller Java application framework", Journal of Computing Sciences in Colleges, 2004
- [13] Paolini P., Garzotto F., Bolchini D., Valenti S., Modelling by Pattern of Web Applications, Int. Workshop on the World-Wide Web and Conceptual Modeling, 1999
- [14] Pastor O., Gómez J., Insfrán E., Pelechano V., The OO-Method approach for information systems modeling: from object-oriented conceptual modeling to automated programming, Information Systems, Special issue on Databases: creation, management and utilization, Elsevier Science Ltd., 2001
- [15] Phillips B., Designers: The Browser War Casualties, Computer, 1998
- [16] Preciado J.C., Linaje M., Pérez J.M., Collado P., PLAG: Plataforma RIA para la gestión centralizada de prácticas de programación, Int. Symp. on Computers and Education, 2005
- [17] Preciado J.C., Linaje M., Sánchez F. and Comai S., Necessity of methodologies to model Rich Internet Applications, IEEE International Symposium on Web Site Evolution, 2005
- [18] Sauter P., Vögler G., Specht G. and Flor T., A Model-View-Controller extension for pervasive multi-client user interfaces, Personal and Ubiquitous Computing, Springer-Verlag, 2005
- [19] Smalltalk MVC: <http://st-www.cs.uiuc.edu>
- [20] Turau V., Web and e-business application: A framework for automatic generation of Web-based data entry applications based on XML, ACM symposium on Applied computing, 2002
- [21] Wojciechowski, J., Sakowicz, B., Dura, K., Napieralski, A., MVC model, struts framework and file upload issues in Web Applications based on J2EE platform, Int. Conference Modern Problems of Radio Engineering, Telecommunications and Computer Science, 2004