A SOFTWARE ARCHITECTURE FOR THE DEVELOPMENT OF COLLABORATIVE WEB APPLICATIONS

Roberto C. Portugal[±], Luis A. Guerrero[†], David A. Fuller[‡]

ABSTRACT

Building collaborative Web applications is not an easy task. In this paper we show a software architecture based on design patterns that allows the organization of collaborative applications on the Web. We described the components that conform this architecture. A prototype of a customizable collaborative virtual environment based on this architecture is also shown. This prototype was implemented in order to validate the architecture.

Keywords: Web-based applications architecture, collaborative Web applications, design patterns.

1. Introduction

The Web architecture is very simple and it is constituted for a client (Web browser) and a server (HTTP server). The Web server just accepts and assists the service requirements that the clients request through the HTTP communication protocol. However, in spite of the simple nature of this architecture, the World Wide Web has become an ideal platform for the development of collaborative applications due mainly to its wide extension around the world and the facilities for the distribution and modification of the applications [9].

The Web nature was basically asynchronous. However, the *last generation Web sites* allows to download HTML documents with Java applets, that provide direct communication, for example through TCP/IP, with other Java servers. These applets can even be on the HTTP server's side like *servlets*, which provides a better performance than the use of the traditional CGIs. These recent advances have made possible the development of collaborative Web applications with a high synchronism degree [4, 7, and 10]. This also solves some of the problems that appear when we try to implement collaborative applications on top of the Web. For example, if we want to show the same information in a different ways to different users, or if we want to make aware the users to any change in the information they have in their browser. However a general architecture that allows collaborative applications to be smoothly integrated into the Web is needed [9].

In this paper we show a software architecture based on patterns, events, and notifications designed to structure the functional components that conform a collaborative application. Through this architecture it is possible to capture the user interaction and to reflect it in the applications interface. It is also possible to manage the shared context of the collaborative applications.

2. The Architecture

The architecture design is based mainly on the *layers* architectural pattern [1], which allows encapsulating the architecture tasks in different levels or modules. Our architecture design also combines the *broker* and *model-view-controller* (MVC) [1] patterns. The *broker* pattern is introduced as intermediate component between the client and the Web server. The broker allows the remote invocation of services. By using the broker pattern we achieve independence between the *client* and *server* components in distributed applications. The client access the server functionality by sending request services through the broker. The broker initiates the communication with the server, sends the requirements and returns the results and exceptions to the clients. This pattern makes possible to encapsulate all server communication routines, as ports definition, communication protocol and server localization. By using the *model-view-controller* pattern we achieved independence among the data maintained by the model component and the application interface. The final architecture is conformed by the *view*, *controller*, *broker* and *model* components. Each component has assigned a very defined functionality. Figure 1 shows the architecture.

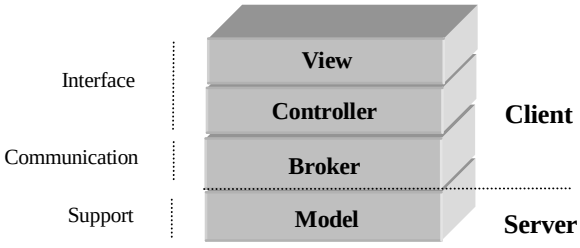

Figure 1. Components of the architecture

[±] Computer Science Department, Pontificia Universidad Católica de Chile. E-mail: rportug@ing.puc.cl

[†] Computer Science Department, Universidad de Chile. E-mail: luguerre@dcc.uchile.cl

[‡] Computer Science Department, Pontificia Universidad Católica de Chile. E-mail: dfuller@ing.puc.cl

The architecture is structured in 3 levels according to the *layer* pattern. The *interface level* is constituted by the view and controller components. The *communication level* implements the broker component, so each application can communicate with the model component. These two levels are implemented in the user's machine (client) inside an Internet browser. The *support level* is in a server machine. This level provides the data storage services.

With this structure we achieved independence between the functionality of the application and the user's interface. Using this model we can show different views to different users, with synchronous updates of any change that occurs in the data. This structure also allows the use of different programming languages for the construction of the client's interface and for the construction of the shared context.

2.1. The Architecture Functionality

The architecture functionality is shown in the following figure. The next sections explain the functionality of each one of the components showed in this figure.

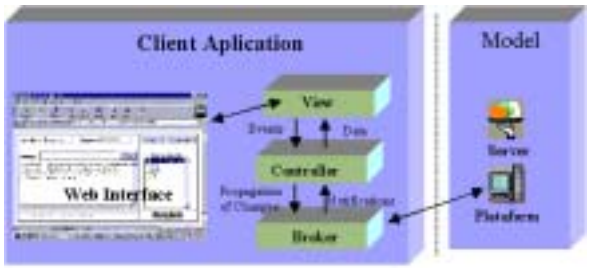

Figure 2. Functionality of the architecture.

2.1.1. The View, Controller and Model Components

The design of the client application is based on an adaptation of the MVC pattern. Through this pattern we separate the data stored in the group memory (the shared context) from the application interface. Due to the independence between the model and the interface in this approach we can implement the user's interface in any programming language.

The interface of a collaborative application is conformed by the *view* and *controller* components. The diagram of the figure 3 shows the behavior of these components. This diagram also integrates the *broker* component in order to explain the notifications behavior, which is detailed later.

The *view* component is responsible to present and to organize the elements of the client's interface, as buttons, text areas, selection areas, menus, and others. This component defines the kind of events that are triggered when the user uses the interface. The actions associated to these events are managed by the *controller* component.

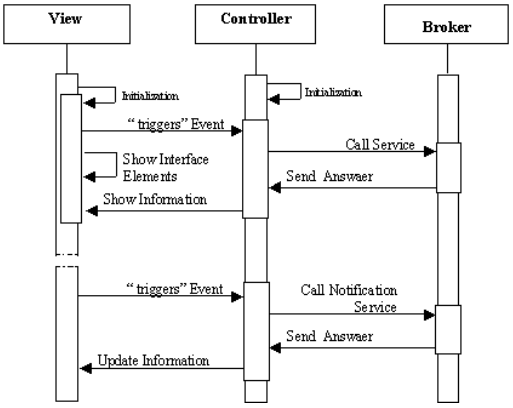

Figure 3. Behavior diagram of the components.

The user interacts with the collaborative application through the *view* component. When the application is downloaded to the client machine through a Web server, the view component shows the structure of the application interface through the elements that compose it. The events are treated by the *controller* component and, in general, imply the model services invocation to obtain the information and to visualize it in the application's interface. During the users' interaction with the elements of the application interface, some events are triggered in the *view* component. These events, as it was already mentioned, are assisted by the *controller* component. This component evaluates the event and determines when it needs to use the *changes propagation mechanism* to maintain consistent the interface of the other users. In the case of an *observer* interface this receives the notification message through the notification service provided by the broker component. The notification message is broadcast toward the controller component, which update the information of the *view* component.

The *changes propagation mechanism* notifies all the users about data changes in the shared environment. Through this service the notifications are propagate to other user applications in order to update their interfaces.

2.1.2. The Broker Component

The distributed applications use the *broker* pattern to abstract the Internet communication aspects between the clients and server. The broker is responsible for coordinating the communication, send requirements and receive the results and exceptions. This pattern provides independence between the client and server components. The client uses the server functionality by sending services request via the broker. The broker sends the requirements to the server, wait for the response, and return the results and exceptions to the client application. The client application does not know anything about the server location, the broker worries about this. The following figure shows the behavior diagram of the broker component.

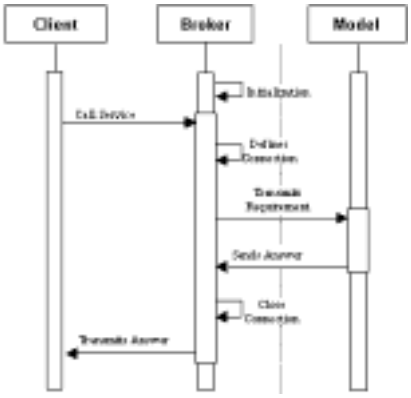

Figure 4. Behavior diagram of the broker component.

The client and broker components are running in the user's machine, while the server component is running in a remote machine. The client and server components have the broker component as the only collaborator.

2.2. The Notifications Service

A common feature in the synchronous collaborative applications is the *notification service*. A *notification* takes place when the state of the shared context or shared environment of an application changes [2]. In our approach, the *model* component provides the collaborative applications shared environment. Through this service the notifications are propagate. We extend the broker component to receive the notifications that the model component propagates and to give them to the client applications [6].

The figure 5 shows the behavior diagram of the *client-broker-server* architecture when the notification service is provided.

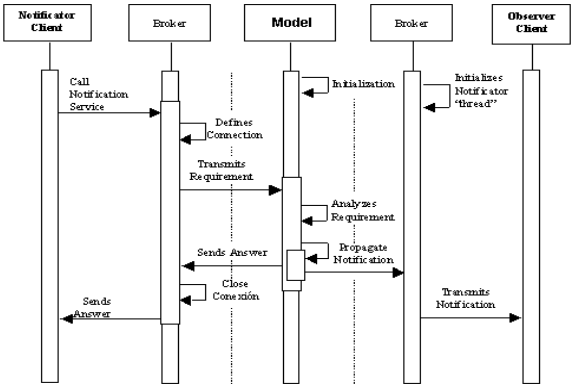

Figure 5. Behavior diagram of the notification service.

There are two non-excluded situations that users can be in a collaborative application: in a *notify* state or in an *observer* state [3]. A *notifier* user produces an event and sends it to the other users. Meanwhile, one user is an *observer* when he or she receives an event notification from a *notifier* user.

3. The DeskTOP Collaborative Application

DeskTOP is a customizable Collaborative Virtual Environment (CVE) built in order to validate the architecture [8]. This application was designed from the perspective of the *desk*, *hall* and *rooms* metaphors and it integrates a set of tools in order to support the interaction among the users in both synchronous and asynchronous way. The users use these tools according to their interaction needs. They can organize the location of the tools inside the desk. Each user can build his own interaction environment with the corresponding tools. Using the DeskTOP tools it is possible to cover the different modalities the people work. Each one of the users participates in the way that he or she feels more comfortable.

DeskTOP provides synchronous and asynchronous collaboration tools and supports the collaboration, communication, and social interaction among the users inside a virtual space in order to achieve a common task. The essence of this CVE resides in that the users are represented inside a shared workspace where they can work freely. They can meet and share information with other users with common interests.

3.1. The DeskTOP Architecture

The DeskTOP system architecture subdivides the model component in two levels. These levels were built using a platform called TOP ("Ten Object Platform") [5, 6]. The TOP platform was used to provide the shared environment or shared context of the DeskTOP application. The TOP platform' server provides services for manage the group memory, the communication aspects, the floor control mechanisms and the work sessions, necessary for the DeskTOP application operation. DeskTOP invokes the services of the TOP server through the *broker* component. The figure 6 shows the DeskTOP system architecture.

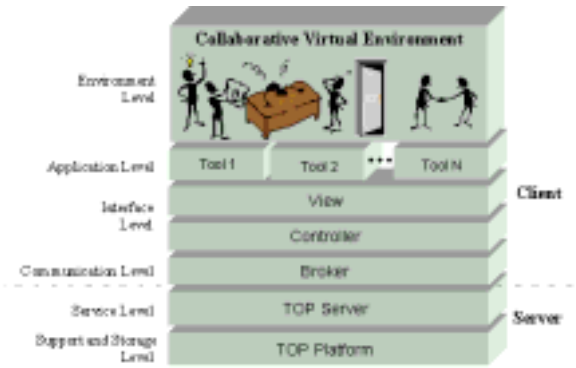

Figure 6. Structure of the DeskTOP system.

Six levels constitute the DeskTOP integral architecture. The *environment* level is composed by a subset of the available tools in the DeskTOP desk. All the tools are available in the *application* level. The *view* and *controller*

components of the interface level are used to build the tools interaction interface. The communication level implements the *broker* component, so each tool can communicate with the TOP server. These levels are implemented at the client's side, while the following levels run in the server machine. The TOP platform server constitutes the *service* level. This server solve the requirements arrived from the tools, which are invoked by the controller component and transmitted by the broker component. The TOP platform is located in the *support and storage* level. This level provides the shared environment of each tool. Figure 7 shows the DeskTOP's system prototype in action.

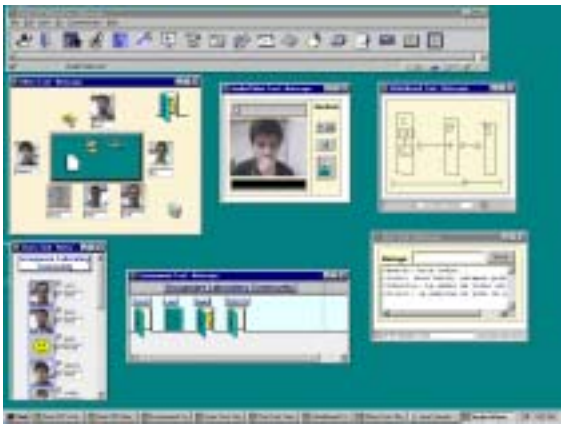

Figure 7. Some users working with DeskTOP.

5. Conclusions and Further Work

In this paper we show a software architecture to structure the functional components of collaborative applications implemented on Web. This architecture was designed based on the *layers*, *broker* and *model-view-controller* patterns. We also show the DeskTOP collaborative application, which was developed to validate the design of the proposed architecture. The implemented *model* and *broker* architecture's components can be reused in the implementation of more collaborative Web applications. The programmer only needs to implement the *view* and *controller* components, which can vary according to the applications.

The proposed focus allows achieving independence between the applications interface and the shared context. The *broker* component provides the necessary communication aspects between the interface and its context. This feature allows taking advantage of the Internet profits for the development of distributed applications, and at the same time to take advantage of the World Wide Web profits for the development of the application' interfaces.

The use of HTML, JavaScript, and Java languages in the implementation of the *view*, *controller*, *broker* and *model* components is appropriate in order to take advantage of the features of each one of these components when we build collaborative Web applications. Using the *layer* HTML tag we achieved a dynamic behavior in the *view* component of the DeskTOP tools. As a future work we need to develop new collaborative applications in order to improve the services of the *model* component. We also need to include cryptography algorithms in the broker component to provide a secure communication service.

Acknowledgments

This work was partially supported by the Chilean Science and Technology Fund (FONDECYT), grant 198-0960.

REFERENCES

[1] Buschman, F., Meunier, R., Rohnert, H., Sommerland, P. and Stal, M, Pattern-Oriented Software Architecture: A System of Patterns, John Wiley & Sons Inc., 1997.

[2] Day, M., Pattersson, J. and Mitchell, D., The Notification Service Transfer Protocol (NSTP): Infrastructure for Synchronous Groupware, Proceedings of the Sixth International World Wide Web Conference, Santa Clara, California, USA, April, 1997.

[3] Eiderbäck, B. and Jiarong, L., A Common Notification Service, Proceedings of the OOGP'97, The ECSCW'97 Workshop on Object Oriented Groupware Platforms, Lancaster, UK, September, 1997.

[4] Gall, U. and Hauck, F., Promondia: A Java-Based Framework for real-time Group Communication in the Web, Proceedings of the Sixth International World Wide Web Conference, California, USA, April, 1997.

[5] Guerrero, L. A. and Fuller, D. A., Objects for Fast Prototyping of Collaborative Applications, Fourth International Workshop on Groupware, CRIWG'98, Rio de Janeiro, Brazil, September, 1998.

[6] Guerrero, L. A., Portugal, R. C. and Fuller, D. A., TOP: A Platform for the Development of Interfaces and Collaborative Application on Web, Proceedings of the XXIV Latin American Conference of Computer Science, CLEI'98, Quito, Ecuador, October, 1998.

[7] Kindberg, T., Mushroom, A Framework for Collaboration and Interaction across the Internet, Proc. of the ERCIM Workshop on CSCW and the Web, Sankt Augustin, Germany, February, 1996.

[8] Portugal, R. C., Guerrero, L. A. and Fuller, D. A., A Customizable Collaborative Virtual Environment on the Web, Fifth International Workshop on Groupware, CRIWG'99, Cancun, Mexico, September, 1999.

[9] Trevor, J., Koch, T. and Woetzel, G., MetaWeb: Bringing Synchronous Groupware to the World Wide Web, Proceedings of the European Conference on Computer Supported Cooperative Work, ECSCW'97, Lancaster, 1997.

[10] Walther, M., Supporting Development of Synchronous Collaboration Tools on the Web with GroCo, Proceedings of the ERCIM Workshop on CSCW and the Web, Sankt Augustin, Germany, February, 1996.