# TOP: A Platform for the Development of Web Interfaces and Collaborative Applications

Luis A. Guerrero
*luguerre@dcc.uchile.cl*
Computer Science Department
Universidad de Chile

Roberto C. Portugal
*rportug@ing.puc.cl*
Computer Science Department
Universidad Católica de Chile

David A. Fuller
*dfuller@ing.puc.cl*
Computer Science Department
Universidad Católica de Chile

**ABSTRACT**

Collaborative applications provide a group of users with the facility to communicate and share data in a coordinate way. This paper shows an object-based platform for the development of Web collaborative applications. A communication outline to exchange messages between a Java server and Web applications is shown. This outline provides a high flexibility for the construction of Web interfaces and collaborative applications on Internet. Examples of Web collaborative applications developed using the object-oriented platform are also shown.

**Keywords:** Collaborative applications, human-computer interfaces, object-oriented platform, Web applications.

## 1. Introduction

The World Wide Web provides an ideal frame for the development of collaborative applications mainly due to its wide extension around the world. This facilitates the distribution of the applications and the communication and collaboration between the workgroup users.

Few years ago the Web nature was basically asynchronous. However, the current Web sites allows to download HTML documents with Java applets providing direct communication to other servers, for example through TCP/IP protocol. The applets can even be at the HTTP server side like *servlets* that provides better performance than the use of traditional CGI's. These recent advances have made possible the development of Web collaborative applications with a high synchronism degree [Bran98, Gall97, Kind96, Trev97, vanW96, Walt96].

However, building collaborative applications is a complex task because it involves issues that do not appear in single-user systems, such as human-to-human communication, group dynamics, users' social roles, group memory, and other organizational and social factors. Most collaborative applications are composed of a set of inter-related modules, including access control, notifications, user management, group interface, packet distribution, data storage, data views, work sessions, awareness information, and user communication [Eide97, Elli91].

In this paper we show a platform for the development of Web collaborative applications and a client-server communication outline. We use as clients HTML documents with JavaScript functions from any browser that support them. Section 2 shows the platform, section 3 the client-server communication outline and section 4 shows some of the applications developed using the platform.

## 2. The TOP Platform

Focusing on the identification of software components for collaborative applications, we studied some tools for the construction of these applications. These tools were: GroupKit [Rose97], NSCA Habanero [Chav98], JSDT ("Java Shared Data Toolkit") [JSDT], NSTP ("Notification Service Transfer Protocol") [Day97], GroCo ("Group Communications") [Walt96], MetaWeb [Trev97], COAST ("COoperative Application System Toolkit") [Schu96], CBE ("Collaboratory Builder's Environment") [Lee96], Artefact [Bran98], and Mushroom [Kind96].

After examining previous development tools, various common concepts were identified that can be applied to the platform components. As shown in Table 1, these concepts are: sessions, users, roles, messages, objects, repositories, and views. The *sessions* (also called *work sessions*, *groups*, or *conferences*) maintain information about the users that interact in an asynchronous or synchronous way through a collaborative application. The *users* (or *collaborators*) are the members of the work group. Every user can have different access rights according to their *roles*. *Messages* (*notifications* or *events*) are used to communicate among users and also among application modules. *Objects* (or *information objects*) are data produced by the users. Most of the time, we need to store objects' attributes, which are information about the information objects (or meta-information) such as the owner of the object, the creation date and time, and previous versions. The objects are stored in *repositories*. Finally, a *view* shows part of the objects in a repository.

| | GroupKit | Habanero | JSDT | NSTP | GroCo | MetaWeb | COAST | CBE | Artefact | Mushroom |
|---|---|---|---|---|---|---|---|---|---|---|
| **Sessions** | X | | X | | X | X | X | X | X | |
| **Users** | X | | X | X | X | X | X | X | X | X |
| **Roles** | X | | | X | X | | | X | X | X |
| **Messages** | X | X | X | X | X | X | X | X | | |
| **Objects** | | | X | X | | X | X | X | X | X |
| **Repositories** | | | | X | | X | X | X | X | X |
| **Views** | | | | | | | X | | | X |

**Table 1.** *Common components in some CSCW toolkits.*

In addition to these seven common features of the collaborative applications toolkits, two more can be included: *environments* and *floor control policies*. Utilizing these nine components, plus a *broker* [Busc96], we constructed a prototype framework for the development of collaborative applications named TOP ("Ten Objects Platform"). TOP is part of the CLASS project ("Computer Learning Applications for Students' Support") [Guer97] and was created to provide tools for the construction and modification of collaborative learning applications. Ten objects for the construction of collaborative applications and a

server that provides access to the methods and attributes of these objects compose this platform. The diagram in Figure 1 shows the relationship among these objects.

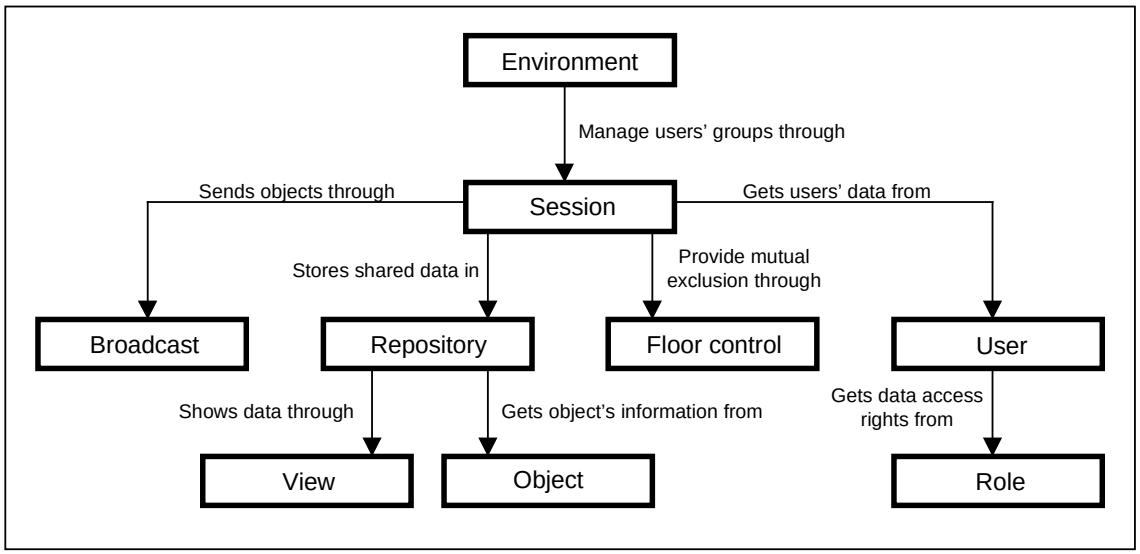


**Figure 1.** *TOP platform objects.*

In TOP platform the *environments* define the general context of the collaborative applications. They allow several workgroups to share an application. Through the *sessions* the environments manage multiple user groups that are working with different data. The sessions can send messages or events to the users through the *broadcast*. The sessions define mutual exclusion on certain objects through the *floor control*. The information about each user in a session is defined in *user*. Each user has certain rights, or data access rights, which are defined through *roles*. The shared information is composed of *objects*. These objects are stored in data *repositories* and they can be seen through *views.*

Through messages sent to the platform server the methods and attributes of all these objects can be modified. Also through this server we can remove and create new objects. The objects and server were built in Java, so the platform is portable.

Ellis defines *groupware* as "computer-based systems that support groups of people engaged in a common task (or goal) and that provides an interface to a shared environment" [Elli91]. Through the objects of the TOP platform we can create the shared environment of the application. The user who builds the applications, through the administrative program of the platform, defines the objects that he or she needs for the application. Then he or she builds the application *interface* using JavaScript. Lets see how we can carry out this process.

## 3. Web interfaces

For the construction of the application interface we use JavaScript and HTML. There is a small applet (the file .class contains 3.11 Kbytes) that encapsulates all the communication routines with the server, including ports definition, communication protocols, notification services and variables for temporary data storage during the time that the application is in the browser. This applet is called from an HTML page that is part of the application, and it allows

the communication with the server of the TOP platform. The main class of this applet is called *TopInterface*. Their attributes and methods are shown in the Figure 2.

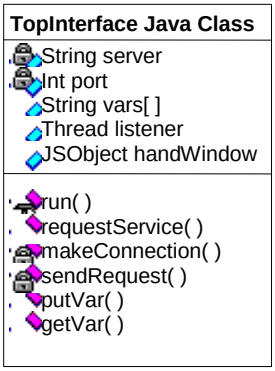


**Figure 2.** *The TopInterface class.*

The attributes *server* and *port* store the name of the machine where the TOP server is running and the socket number that it uses to listen to the messages. The attribute *listener* is a reference to a thread object. The *handWindow* is an attribute that represents the browser window (or frame) where TopInterface is executed. *handWindow* provides the communication connection between TopInterface and the JavaScript functions.

A common problem in JavaScript language is that we cannot use variables from functions that are executed in different frames of a browser. In order to solve this problem, TopInterface has the attribute *vars[]* that is a strings vector with limited capacity. We can use this attribute through the methods `putvar()` and `getvar()` that store a variable and recover its content, respectively.

The method `run()` receives the notifications from the server and transfers them to the JavaScript function `notificator()`. This method is a protected thread that is executed after having initialized and set the attribute *listener*.

```
listener = new Thread(this);
listener.start();
```

`requestService()` is a public method for the JavaScript functions that want to communicate with the TOP server. `makeConnection()` is a private method that establishes this communication via sockets. Finally, `sendRequest()` is the private method that sends messages to the server. These messages are originated in the JavaScript functions of the HTML application pages.

### 3.1 TopInterface Functionality

*LiveConnect* is the mechanism that allows JavaScript and Java to work together. We use this communication facility to achieve the interaction among the Web applications with JavaScript and the TopInterface applet.

Two characteristics that TopInterface offers to the JavaScript applications are the communication with the TOP platform (for the invocation of their services) and a notification service to provide user awareness [Dour92]. The Figure 3 shows this functionality.

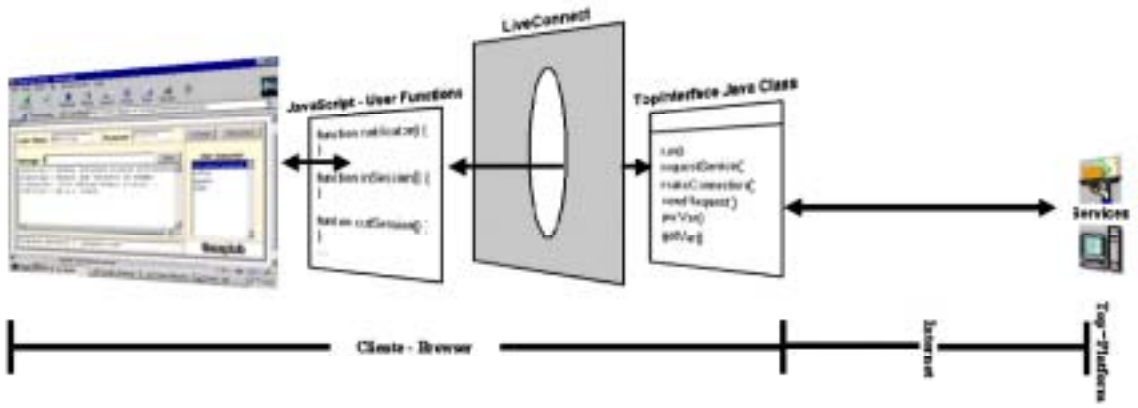


**Figure 3.** *TOP Server Communication process.*

The events that happen in the interface (browser) are handled by the respective functions defined in JavaScript. For example, when a user presses a button to be connected to a specific session, the function `inSession()` is invoked. This is shown next.

```
<INPUT TYPE="button"
      NAME="button_connect"
      VALUE="Connect"
      onFocus='document.formData.msg.value="Connect to a TOP session."'
      onClick="inSession(document.formData.text_userName.value,
                         document.formData.text_password.value)">
```

This JavaScript code defines a button to be connected to a work session of the TOP platform. The event that handle this connection is the function `inSession()`, using the user's name and password.

### 3.2 TOP Server services

When we need to invoke a service from the TOP platform the respective message should be built in a JavaScript function, according to the structure of the TOP services. For example, when a user wants to be connected to a TOP session he should invoke the following TOP service:

```
UpdateUserSession:user_name,password,session_name,app_name,check_user,acces
s_mode
```

For this, he should build in a JavaScript function the following message:

```
      var msg = "UpdateUserSession:" + userName + "," + password + "," +
                  sessionName + "," + AppName + "," + "not" + "," + "in";
```

To transmit this service requirement the method `requestService()` is invoked. This method is transparent for the users, and it is carried out in the following way:

```
answer = document.TopInterface.requestService(msg)
if (answer=="OK") {
      // Service successful
}
else {
      // Problems in the service
}
```

The variable *answer* receives from TOP the confirmation of the service. When the service is successfully assisted, an "OK" is received. Otherwise, an "ERROR" code is received. The method `requestService()` receives the message in the variable *service*.

```
// Public method for JavaScript applications
public String requestService(String service) {
      String answer="";
      try {
            makeConnection();
            answer = sendRequest(service);
      } catch (IOException ex) {
        ex.printStackTrace ();
      }
      return(answer);
}
```

The method `makeConnection()` takes care of establishing the communication with the TOP server via sockets, and the message is sent through the method `sendRequest(service)`. This method waits the server confirmation and returns it to the JavaScript function in the variable *answer*.

### 3.3 Notification Services for Web Applications

A common feature in synchronous collaborative applications is the *notification service* to the users. A *notification* takes place when the shared state of an application changes [Day97]. Through the notification service that TopInterface has we can provide awareness and synchronism mechanisms.

There are two non-excluding situations that the users can be in a collaborative application: in *notify* state or in *observer* state [Eide97]. A user is a *notifier* when he produces an event and sends it to the other users. For this situation we use the method `sendMessage()` of the TOP object *broadcast*. We built the message for this service in a JavaScript function and then we invoked the method `requestService()` in order to transmit the message to the server. For example:

```
msg = "broadcast:" + userName + "," + appName + "," + sessionName +
      "," + broadcastName + "," + userList + "," + "nosave" + "," +
      messageToBroadcast

answer = document.TopInterface.requestService(msg)
```

A user is an *observer* when he receives from a *notifier* user an event notification. TopInterface implements a thread at the client's side to listen the messages transmitted by the notifier users. This is carrying out through the service `broadcast()`. This situation force to implement in JavaScript an event handler function named `notificator(String notify)`. This function is invoked by the TopInterface thread to transmit the notification through the attribute *handWindow* in the following way:

```
handWindow = JSObject.getWindow(this); // handler for Netscape window
...
private void run() {
      ...
      handWindow.call("notificator",args);
      ...
}
```

The attribute *handWindow* makes reference to the Web page where TopInterface is being executed. Through its method `call()` the JavaScript function `notificator()` is invoked. The argument *args* contains the event to notify. In the construction of the interface the handling of all the events that can produce notifications should be considered. This depends on the functionality of each application.

The notifications are received by the function `notificator(String notify)` in the variable *notify*, under the following format:

| Kind of Event | Parameters |
|---|---|

Some examples of notifications that we could require to manage are:

- A user arrival to a work session
- A user departure from a work session
- A message reception from a user
- A change in a telepointer position
- A change in the URL address of an HTML page
- An object sends to a repository by a user

Let see some examples of Web applications that use the TOP platform objects and the communication outline.

## 4. Application Examples

The following applications were implemented on Web using JavaScript. These applications use the class *TopInterface* and other objects of the TOP platform to provide collaboration.

For the creation of the *environments* of each application we use the administrative program of the TOP platform. Through this program we can create and modify all the platform objects. The main screen of this program is shown in Figure 4.

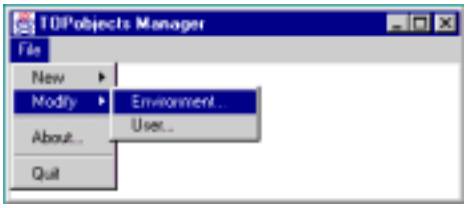


**Figure 4.** *TOP platform objects administrator.*

Through the main menu users and environments can be created. Once we create an environment, the other related objects can be created. In a previous stage of the applications design we should determine the objects that will be needed and through this administrator we create them. Let's see some examples.

### 4.1 *ChatWeb* Application

A chat application was implemented to demonstrate the functionality of our platform in the construction of Web collaborative synchronous applications. This application sends and receives text messages and provides awareness mechanisms over the users that arrive and depart from the session. The messages are sent to all the connected users in the session. However, the application provides the facility of selecting the users we want to send the messages.

For this application we create in the TOP platform (using the administrator showed in the previous section) a work *environment* and a *repository* with *broadcast* without persistence. We defined a guest *user*, thus any person can use this application. The Figure 5 shows our chat application.


**Figure 5.** *Chat through the Web using TOP platform.*

The user should enter his name and the word "guest" as password. Then he should be connected to the platform through the button "connect". Once registered in the platform he can send messages. If other users enter or leave the session, they will be visualized in the window "Users Connected". To leave the session the button "disconnect" is used.

### 4.2 *Slider* application

Several related works exist on the synchronization of the Web Browsers [Yeh96, Parn97]. *Slider* requires the creation of a *session*, a *repository* and a *floor control*. With the floor control we guaranteed the existence of a single notifier user and several observer users. The user with the floor control has enabled the toolbar in order to navigate inside a sequence of HTML pages. Also he has a *telepointer* [Gree96] implemented with *layers* [Netscape-Layers] to focus some topics in the HTML pages.

The contribution of Slider resides in its implementation. According to Yeh [Yeh96], JavaScript and Java cannot synchronize the browsers in the clients: JavaScript does not provide communication capabilities and Java only detects events that are activated in the frame of the Java applet. Slider syn-chronizes the browsers of the clients through JavaScript functions that communicate with Java applets.

The problem of offering real time distribution among HTML pages can be divided into two parts: *synchronization* and *distribution* [Parn97]. The synchronization and distribution in the Slider application is given by the JavaScript function `notificator()`. This function makes that the observer users receive two types of notifications: the position of the telepointer and the URL address of the HTML pages. With the URL address the observers request the HTML page to the Web Server. With the telepointer position they draw the telepointer icon in their own browser. The Figure 6 shows the Slider application.


**Figure 6.** *Slider application.*

When loading the main HTML page the user's name is requested. The application automatically detects, according to the configuration of the TOP work session, if he has or not the floor control. Any action that is carrying out by the user who has the floor control (on the

telepointer or the navigation buttons) is reflected in all the connected browsers to the session. At this moment only the coordinator user has the floor control. We are working in a new version that allows passing the floor control using some politics as FIFO or priorities.

### 4.3 ECO: An Asynchronous Collaborative Editor

ECO is a simple asynchronous collaborative editor, built using our communication outline and the TOP platform. The access to the editor is restricted. Only previously defined users can create or modify documents. When entering to the editor the user's name and password is requested. Through a JavaScript function, a message is sent to TopInterface to validate these data. TopInterface communicates with the TOP server to verify if he is an editor's user. If the user has access to the editor the Figure 7 screen is showed.

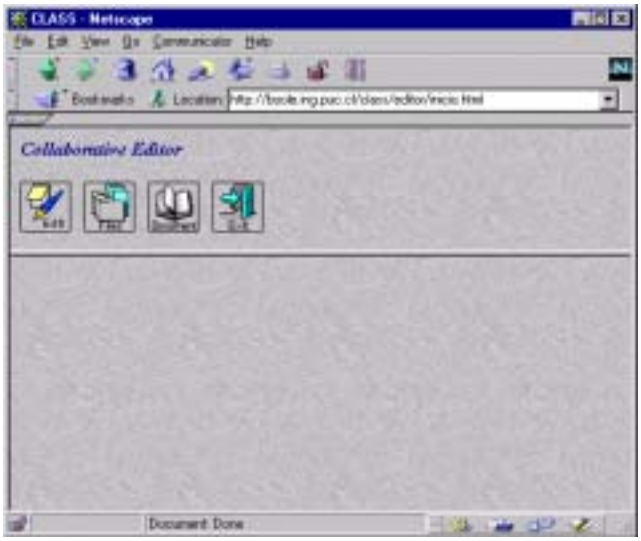


**Figure 7.** *Asynchronous collaborative editor.*

In this screen we have options for text edition, create or recover documents and visualize the documents. According to the defined collaboration strategy the users cannot erase neither to modify objects created by other users. They can create new objects, for example new text paragraphs, or modify those objects that they have created. However, they can make comments to any object through text notes. The owner of the object can remove these notes. The Figure 8 shows an example of document visualization including these comment notes.

**Figure 8.** *Comments notes in ECO editor.*

The figure 8 shows the option for visualize the document with all its marks. Here is included, in blue text, the name of each object, the owner and the icons corresponding to the comments made by other users. These comments can be visualized by clicking over the icons. In order to show the comments a small yellow window is showed with the name of the user that made the comment and the text. Another option shows the complete document without marks, in its final presentation.

## 5. Conclusions and Further Work

In this paper we show a platform for the construction of Web collaborative applications and a communication outline among JavaScript functions, a Java applet, and a Java server, based on the *broker* design pattern.

The *repository* and *object* components of the TOP platform manage the data of the collaborative applications. The *view* object provides different views of those data. The *user* and *role* objects define the user's access rights over the views and the information contained in the repositories. The *floorControl* object provides coordination in the use of shared resources. The *session* object allows defining periods of synchronous work. It also defines different work areas for different work groups. The *broadcast* object distributes objects among the users connected in a synchronous work session. Finally, the *environment* object establishes the relationship among all the objects of a single application. The methods and attributes of these objects are administered through a Java server.

The *environment* component allows several user groups to simultaneously share an application without interfering with others groups. This allows the transformation of *mono-group* applications into *multi-group* applications. The *environment* and *session* components allow re-use of the applications, as it is sufficient enough to create new sessions for other groups to use the applications

Some of the main features of collaborative applications can be designed from these components, including, for example, shared data (*repository*, *objects*, *session*), communication among users (*broadcast*), coordination (*floor control*, *users*, *roles*), group memory (*repository*, *objects*), users (*users*), roles (*roles*), data awareness (*objects*, *repositories* with versions), user awareness (*users*, *session*s), data views (*views*), work sessions (*session*), access control (*roles*, *sessions*), floor control mechanisms (*floor control*), notifications (*broadcast*), events, and messages (*broadcast*).

An environment or shared context and an interface to this environment compose a collaborative application. Through the ten objects of the TOP platform is possible to build the environment or shared context. For the creation of the Web interface of the collaborative applications we use the JavaScript language. The JavaScript functions can communicate with the TOP server through the TopInterface applet using *LiveConnect*.

The TopInterface applet has a method that "listens" the messages sent by the server. When receiving a notification, it executes an events handler that should be programmed in the Web application like a JavaScript function named `notificator(String notify)`. Through this function each application makes the respective treatment of the notifications that it can receive. Another functionality of TopInterface is the storage of JavaScript variables, avoiding the use of "cookies" that can even be disabled for the user of the browser. This solves the problem that JavaScript, for security reasons, does not allow transferring information among frames.

Our proposal shows how Web applications with JavaScript functions can be extended to synchronous, multi-user and distributed applications, using TopInterface and the TOP platform. Allowing then adding collaboration mechanisms to these applications.

Although in this paper we have emphasized in the development of Web collaborative applications, our platform is an open platform. It was built in Java, so it is portable. The platform listens to a port defined by the administrator when he runs the server. It is possible the communication among the server and any application built in any language that allows sockets definition.

As future work we expect to design and to implement more collaborative applications using the platform. We are working in the incorporation of audio and video tools, allowing a better user communication using *plug-ins* through *LiveConnect*. We also expect to incorporate more services in the TOP server, as well as to improve the aspects of information security and consistency.

## Acknowledgments

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

## References

[Bran98] Brandenburg, J. et al. “Artefact: A Framework for Low-Overhead Web-Based Collaborative Systems”. Proceedings of CSCW'98, Seattle, Washington, 1998.

[Busc96] Buschmann, F. Meunier, R., Rohnert, H., Sommerlad, P. and Stal, M. “Pattern-Oriented Software Architecture: A System of Patterns”, John Wiley & Sons, 1996.

[Chav98] Chavert, A., Grossman, E., Jackson L., Pietrowics, S. and Seguin, C. “Java Object-Sharing in Habanero”. Communications of the ACM, Vol. 41, No. 6, June 1998, pp. 69-76.

[Day97] 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.

[Dour92] Dourish P. and Belloti V. “Awareness and Coordination in Shared Workspaces”. Proceedings of CSCW'92, pp. 107-114. Canada, 1992.

[Eide97] 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 7, 1997.

[Elli91] Ellis, C.A., Gibbs, S.J. and Rein, G.L. “Groupware Some Issues and Experiences”. Communications of the ACM, Vol. 34 No.1, 1991, pp. 38-58.

[Gall97] Gall, U., Hauck, F. “Promondia: A Java-Based Framework for Real-Time Group Communication in the Web”. Proceedings of the Sixth International World Wide Web Conference. Santa Clara, California USA, April, 1997.

[Gree96] Greenberg, S., Gutwin, C., and Roseman, M. “Semantic Telepointers for Groupware”. Proceedings of the OzCHI '96, Sixth Australian Conference on Computer-Human Interaction, Hamilton, New Zealand, November, 1996.

[Guer97] Guerrero, L. and Fuller, D. “CLASS: A Computer Platform for the Development of Education's Collaborative Applications”. Proceedings of Third Internatonal Workshop on Groupware, CRIWG'97, Madrid, Spain, October, 1997.

[Kind96] Kindberg, T. Mushroom, “A Framework for Collaboration and Interaction Across the Internet”. Proceedings of the ERCIM Workshop on CSCW and the Web, Sankt Augustin, Germany, February, 1996.

[Lee96] Lee, J., Prakash, A., Jaeger, T. and Wu, G. “Supporting Multi-User, Multi-Applet Workspaces in CBE”. Proceedings of CSCW’96, Boston, Massachusetts, USA, 1996.

[Parn97] Parnes, P., Mattsson, M., Synnes, K. and Schefstr, D. “The mWeb Presentation Framework”. Proceedings of the Sixth International World Wide Web Conference. Santa Clara, California USA, April, 1997.

[Rose97] Roseman, M. and Greenberg, S. “Building Groupware with GroupKit”. In M. Harrison (Ed.) Tcl/Tk Tools, pp. 535-564, O'Reilly Press, 1997.

[Schu96] Schuckmann, C., Kirchner, L., Schümmer, J. and Haake, J. “Designing object-oriented synchronous groupware with COAST”. Proceedings of CSCW’96, Boston, USA, 1996.

[Trev97] 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.

[vanW96] vanWelie M. and Eliëns A. “Chatting on the Web”. Proceedings of the ERCIM Workshop on CSCW and the Web, Sankt Augustin, Germany, February, 1996.

[Walt96] 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.

[Yeh96] Yeh, P., Chen, B., Lai, M. and Yuan, S. “Synchronous Navigation Control for Distance Learning on the Web”. Proceedings of the Fifth International World Wide Web Conference, Paris, France, May, 1996.

## URL References

[JWS] Java Web Server. http://jeeves.javasoft.com

[JSDT] Java Shared Data Toolkit.
http://www.javasoft.com/products/javamedia/jsdt/index.html

[Netscape-Layers] Layers and JavaScript Extensions for Layers Glossary.
http://home.netscape.com/comprod/products/communicator/layers/layers_glossarl