Objects for Fast Prototyping of Collaborative Applications

Luis A. Guerrero, David A. Fuller
{luguerre, dfuller}@ing.puc.cl
Computer Science Department
Pontificia Universidad Católica de Chile

Abstract

This paper presents an objects-based framework for building collaborative applications. The framework consists of ten objects: *box*, *boxObject*, *environment*, *user*, *role*, *session*, *broadcast*, *view*, *boxObjectType* and *floorControl*. We argue that we can build a big number of collaborative applications, including Web applications, by combining these objects. The main aspects in the design of the framework and each object are discussed, as well as the designing and building of collaborative application process using the framework.

Keywords: Collaborative applications construction, OO framework, collaborative Web applications.

1. Introduction

For some time our group has been working on the technology support to the teaching-learning process in our university. We created a project called CLASS ("Collaborative Learning Applications for Students' Support") [Guerrero97] to achieve this goal. This project has two main goals: the creation of collaborative environments that support interaction among students, assistants and lecturers, mainly based on Web, and the creation of "flexible" Web documents that allow information queries and different views according to the users' role. Most of the work in this project is focused on Web since this is already a very successful tool to build collaborative environments [Dix96], and also an easy access platform for the universities.

From the beginning of the project we observed that we need a simple but powerful tool for the fast construction and modification of collaborative applications. It was the beginning of TOP ("Ten Objects Platform"), an objects-based framework for the construction of collaborative applications, mainly on top of the Web. This platform is the one we present in this paper.

At present, there are few projects that have similar goals. The Habanero project at NCSA [Habanero] is a framework for sharing Java objects with colleagues distributed around the Internet. However, it does not provide tools for managing data or facilities for the development of Web applications. Habanero is a good framework for the development of Java applications, but it does not have facilities for building applications in other computer languages.

BSCW ("Basic Support for Cooperative Work") [Pieper97] enables collaboration over the Web. BSCW is a "shared workspace" system, which supports document uploading, event notification, group management and data awareness and version control. However, it does not provide tools to manage data nor facilities for the development of collaborative applications.

Web-CT ("World Wide Web Course Tool") [Goldberg96] is an easy-to-use environment for creating WWW-based courses that are otherwise beyond the ability of the non-computer programmer. However, it does not provide collaboration facilities or data administration.

NSTP ("Notification Service Transfer Protocol") [Day97] is a good infrastructure system for building synchronous groupware. However it lacks facilities for the construction of asynchronous collaborative applications.

GroupKit [Roseman96] is a very good tool for building real-time applications such as drawing tools, editors and meeting tools that are shared simultaneously among several users. However, it does not provide facilities for the construction of asynchronous collaborative applications.

The TOP framework contains the facilities of the above-mentioned systems and more. We can share objects through a Web interface and we have data administration tools, object persistence and facilities for the construction of synchronous and asynchronous collaborative applications, including collaborative Web applications. TOP framework objects were designed keeping in mind some of the main groupware characteristics.

2. The TOP Framework

Three steps should be followed when building and designing the applications over TOP: identification of the needed objects; objects creation and building process of the user interface. This interface can be build in any language that allow us to define sockets, or else through JavaScript [Wooldridge97] or Java applets [Flanagan97] if we are working over Web. The objects creation is made by an application built for the TOP framework administration. In order to identify the objects needed on each application we need to know each object in more detail.

2.1 The *box* object and the *boxObject* object

One of the main design characteristics of the platform is the group memory [Ellis91, Conklin92]. This memory stores all the information generated during the group working sessions. The final product of the workgroup will also be stored here, as well as all other information for user and data awareness [Dourish92]. For the group memory administration we use the *box* object.

A *box* object is an information repository in which other objects can be kept. For each object kept in a box, the framework stores specific information. This information is stored in another object called *boxObject*. Therefore the *boxObject* maintains administrative information on all the objects that are contained in a *box*.

Each *box* object contains the following information: name, owner or creator, description, state (open, closed, erased, system), versions (yes or no), types of objects that accepts, users list and access rights.

According to the state, a box can be *open*, which means in this case that the users can put in and take out objects; *closed*, when it does not allow objects to be put in or taken out; logically *erased*, a state in which only a framework administrator can recover it; and *system*, which is a special state of the boxes that belong to the same framework.

Awareness mechanisms play a very important role in collaborative applications because they give to the users the feelings that are working in groups. In order to provide awareness mechanisms, we should give to the users some information about what other users are doing or have done recently. All this information is stored, in some way, in the group memory. In order to help the applications to show awareness information to the users, the boxes have the option of keeping objects versions. When this option is activated in a box, the objects are not physically erased or updated. Each object maintains a list with its previous versions.

It can also be indicated what type of objects a box can accept. The object types are defined inside the same platform. When a user tries to put an object inside a box, it first checks if this object type is accepted. It also checks to see whether the box is open, and if the user has enough access rights over that box. Each box has a list of roles with the following rights: to *put* (P), to *erase* (E), to *modify* (M) and a special right called *owner* (O) that grants all of the rights over an object if the user is the owner of that object.

The *box* object also has methods to managing its information. For example, just to name a few, there are methods for putting objects, for taking them out, for asking if an object exists inside a box, for giving a list of all the box objects or a list of a specific object type, for recovering a number of previous versions of an object, for closing a box, for opening it and for erasing it.

There is an instance of *boxObject* for each object inside a *box*. This object contains, among another information, the owner of the object, the creation date, the last modification date, the previous versions list (if the box accepts versions), and the keywords. The object keywords will allow us to make queries about the information of the box.

2.2 The *environment* object

Ellis said that Groupware are computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared *environment* [Ellis91].

The *environment* object contains information about the whole application environment: its name, description, owner or administrator, users and roles list and the boxes list. Through the *environment* object, an application has access to its work area. All the other framework objects must be associated to an environment.

This object has methods to create, erase and modify environments; to ask if a specific environment exists; to supply a list of existing environments; to ask if a user is member of the environment; to ask users access rights; to ask for the boxes associated with the environment; and to ask which sessions are associated with that environment.

2.3 The *user* object

The *user* object contains information about the users of the framework: their login names, passwords, complete names, email addresses, photographs, addresses, phone numbers and last date and time each user worked on a framework object.

This object has methods to create, erase and modify users, to ask if a user exists, to supply a user list and to return information about a specific user.

2.4 The *role* object

A *role* is a set of privileges and responsibilities attributed to a person [Ellis91]. The *role* object contains the access rights of a set of users over the objects of a box or a view. These rights can be: to put objects, erase objects, modify objects and/or see objects. It has methods to create, erase and modify roles.

2.5 The *broadcast* object

The *broadcast* object is an extension of the *box* object. A *broadcast* is a special box type that maintains a connected user list for a session and their respective host and port numbers. Any object sends to a broadcast object is automatically sends to all the users on the *broadcast* list. The *broadcast* object has an option for store the objects sent to it. It also has information about the type of objects on which the broadcast should be performed, and the port number to which the objects should be sent.

This object has methods to create, erase and modify *broadcast* boxes, to send an object to the *broadcast* box, to include new connected users to the session and to remove users from the broadcast session.

2.6 The *session* object

A *session* is a period of synchronous or asynchronous interaction supported by a groupware system [Ellis91]. The *session* object allows different work sessions to be defined inside the same environment. This object matches the boxes defined in the environment according to the user's name and the session in which he is working. This object allows different workgroups to use the same application, without making any change in the application. In this way, we can create collaborative applications for a workgroup and allows that many groups use these applications simultaneously. The *session* object extends single-group applications to multi-group applications.

This object contains an identification, description, name of the environment it belongs, user list, connected user list (including their host names and port numbers), boxes list, date and time for the beginning and ending of the session, and a coordinator.

It has methods to create, erase and modify sessions, to ask if a user is a member or coordinate a session, to ask if a user is connected to the session, to check if the session is open (according to the beginning and ending time) and to ask the host name and the port number of a connected user.

2.7 The *boxObjectType* object

The *boxObjectType* object allows defining new types of objects that will be stored by the boxes. For each type this object defines a name, an extension, an icon and the respective visualization information for the particular object type. For example, if there are defined graphic objects for a Web application, the object type should contain in its visualization tag something like `<IMG SRC=<PATH> BORDER=0>`. Using this tag, the system knows the way in which it should present the objects when being visualized. In these tags the constants `<PATH>` and `<OBJ>` can be used. The complete path of the object will automatically substitute the first one (for example when displaying graphics) and the complete object (for example when visualizing text) will substitute the second one. Icons can also be assigned to the object types. This allows a graphic visualization of the types in many situations such as when showing lists of objects.

This object has methods to create, erase and modify types, to ask if a type exists, and to return the icon, tags or extension of a specific type.

2.8 The *view* object

A *view* is a visual or multimedia representation of some portion of a shared context [Ellis91]. The *view* object allows different views to be defined on the same objects of a box. A *view* can be of three types: *html (H), list (L)* and *query (Q)*. "Html" when the view is a simple HTML [Berners-Lee94] documents, for Web applications; "list" when an orderly list of objects defines the structure of the view. For example, if we have a box to store a document, the first object in the list will probably be the document title, the second object the authors, then the abstract, the introduction, etc. For a great number of applications it is enough to have an orderly list of the objects for the views. The view type "query" is the most complex, and based on a very simple SQL language, it allows us to make queries on the objects of a box. For example, queries like: "select all the exercises from chapter 2 that deal with pointers". This would entail placing a select on the box named "chapter 2" of the "exercise" objects that contains the word "pointer" among their keywords.

The *view* object contains a `refresh()` method that builds and stores a static view. For example, for Web application `refresh()` create the html document corresponding to that view or query. This document is stored and displayed every time a user wants to see this view. The SQL queries are executed only a few times, when the information of the box is changed. Each *view* maintains a list of roles.

It also has methods to create, erase and modify views, to update the order of the object list, to supply the list of views of a box, to see the view as a HTML page and to see the object list of the view (using the corresponding icons for each type).

Through the view objects we can create HTML documents that uses hypertext, graphics, audio and video. However, the boxes can also stored HTML pages with all these elements.

2.9 The *floorControl* object

The *floorControl* object allows us the assignment of a *floor passing protocol* [Ellis91] to an object type in a box. When an object has floor control, only a user can have that object at a certain moment. If another user requests the same object, he is added to a waiting list according to the floor control policy requested when creating the floor control (FIFO, priorities, etc.). When the user that has the object releases it, the object is assigned to the following user in the list. For example, if we want to create an application with only one telepointer object [Greenberg96], we need to assign a floor control to it. Using a *broadcast* object, we can have an application where one at time users request the telepointer to make indications to the other users, or to write on a whiteboard.

This object contains an ID, the name of the associated box, the name of the user that has the floor control and a queue for the users who are requesting the floor control. It has methods for requesting and releasing floor control, for assigning it and for placing users in queues.

3. Applications Design

As we mentioned in the previous section, the first step in the design of the applications is to identify the objects that will be needed. An application built over the TOP framework needs to define its work environment. Once we define the environment, we can define the users and their roles. Then it is necessary to structure the group memory, in order to define how many repositories or boxes we need for the application. We also decide if the boxes will maintain versions (in order to provide data awareness mechanisms). Once the boxes are identified, we can define the object types that will be stored in those boxes. Then we define the different views on the boxes. Finally we add, in case the application requires it, the floor controls (coordination), the broadcast boxes (communication, synchronous applications, notifications) and the work sessions.

If we need to re-use applications already created, it is necessary to create new sessions for those applications and to incorporate these sessions to the new work environment. The session object also allows that several users groups use the applications in a simultaneous way, without interfering among each other's.

Let's see some of the main characteristics of the collaborative applications (according to Ellis et al.) and the TOP objects needed to implement them.

Groupware characteristics	***Objects in TOP framework***
Collaboration	All the objects.
Communication	*broadcast* (synchronous), *box* (asynchronous).
Coordination	*floorControl*.

Shared context	*environment, user, box, boxObject*.
Group memory	*box, boxObject, boxObjectType, view*.
Users	*user*.
Roles	*role*.
Awareness	*box with versions, session, user, environment.*
Views	*view, box, boxObject, boxObjectType*.
Sessions	*session*.
Floor control	*floorControl*.
Notifications	*broadcast.*
Access control	*role*.
Group interface	*TopInterface* applet for Web applications.

Table 3.1. *Groupware characteristics and the TOP objects that support them.*

The TOP platform allows defining and managing the shared context or shared environment of the collaborative applications. However, the group interface should be built using some conventional computer language. For Web applications, we built an applet named *TopInterface*, which encapsulate all the communications routines. Other human-computer interface related aspects such as graphical representations, telepointers [Greenberg96] or shared windows [Ellis91] should also be built in some computer language. For notification services [Patterson96], we use the broadcast object.

The TOP objects are available through a main server. The applications should send their messages to this server in order to use the objects' methods. The following figure shows the communication outline between the applications and the TOP objects' server.

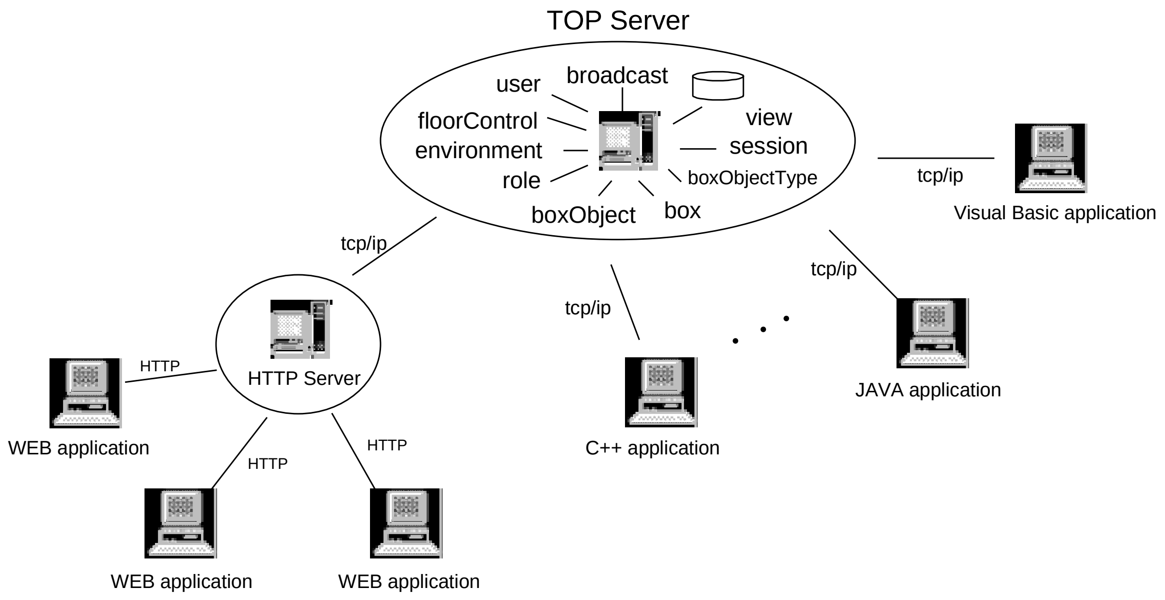
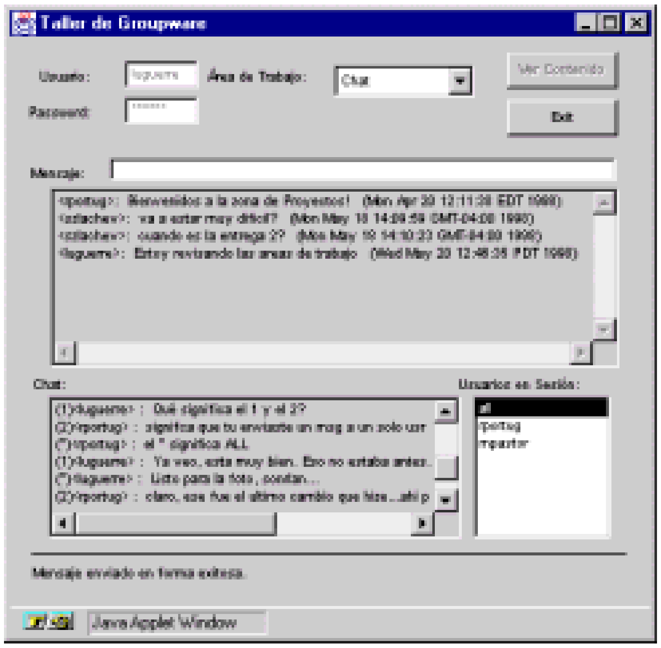


Figure 3.1. *TOP objects, server and applications interfaces.*

In order to build a TOP application interface we need a computer language that allows defining sockets to communicate with the server. Using this language the human-computer interface is programmed and messages are sent to the server to execute the methods of the corresponding objects. To create and manage the objects, we use a Java application. Let see an example of an application built on TOP.

4. Discussion Groups and Chat Application

This application was developed as a Java applet loaded from an HTML page. The main objective of this application is to provide discussion areas to the students, assistants and lecturer of the course "Groupware". Several course topics are shown by the application and the users can send comments to each one. Some topics are "officials" for the course, only the lecturer or the assistants can send comments to them. The application also has a chat window [vanWelie96] with a list of connected users in the session. We can send messages to all the connected users, to a single user or to a group of users selected with the mouse. The users can read the comments of the different interest areas, and can use the chat to talk or to discuss some topic with other users. The following figure shows this application.



Usuario: lfuguerro

<rportug>: Bienvenidos a la zona de Proyectos! (Mon Apr 20 12:11:38 EDT 1998)
<zdiachev>: va a estar muy dificil? (Mon May 18 14:09:59 GMT-04:00 1998)
<zdiachev>: cuando es la entrega 2? (Mon May 18 14:10:23 GMT-04:00 1998)
<lfuguerro>: Estoy revisando las areas de trabajo (Wed May 20 12:48:35 PDT 1998)

(1)<lfuguerro> : Qué significa el 1 y el 2?
(2)<rportug> : significa que tu enviaste un msg a un solo usr
(*)<rportug> : el * significa ALL
(1)<lfuguerro> : Ya veo, esta muy bien. Eso no estaba antes.
(*)<lfuguerro> : Listo para la foto , contan...
(2)<rportug> : claro, ese fue el ultimo cambio que hice....ahi p

all
rportug
mpastor

Mensaje enviado en forma extensa.

Figure 4.1. *Discussion groups and chat*

For this application we first create an *environment* object. Then we create four *users*: students, assistants, lecturer and guest. We create a *box* for each interest topic and a *broadcast* box for the chat. We build an "orderly list" type *view* object for each *box*. Then we define the user’s roles according to the boxes: the students cannot send objects to the official boxes. The guest users can only use the chat, but they do not have access to the interest areas’ boxes. We use a session for the whole semester the course is taught. Finally we build a "comment" type object (*boxObjectType* object) that can store the messages in the *boxes*.

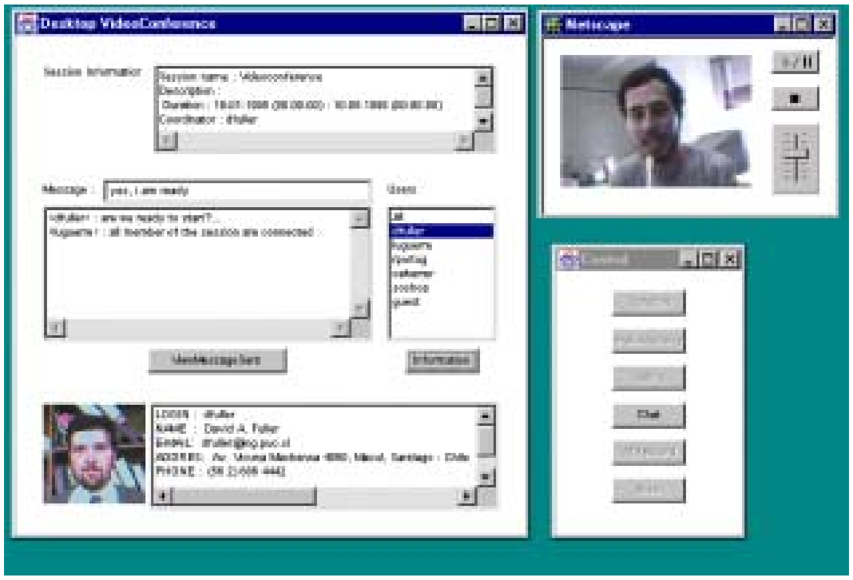
The interface was built in Java. When entering the name and the user's password, the application sends a message to check if the user has access to enter to that session. Once validated, the interest area (*boxes*) or the chat (*broadcast*) must be selected. After writing a message and typing an ENTER, a new "comment" object type is created and it is sent to the respective *box* or *broadcast*. If we send a comment to the broadcast, this will appear in the screens of all the users connected in the session. If we send a comment to an interest area, the comment is stored in the respective area *box*. If we want to see all the sent messages, the *view* object associated to the respective *box* is used. When choosing the leaving option, a message to disconnect to the session is sent.

5. An Electronic Meeting System

This application uses the TOP framework to coordinate the shifts (through a floor control object) of video and audio transmission among the participants of a meeting. For the audio and video capture, transmission and visualization we used commercial software. The system allows a group of people to participate in a distributed synchronous meeting, with two possible options: audio and video transmission or text-only transmission. The application presents a screen with options to request the floor control for audio and video. It also presents a chat window where a user can send a message to the whole group or to some specific user, selecting from a list of users who are participating in the session.

In this application we define two *roles*: meeting coordinator and participants. The coordinator can participate in the meeting just like any other user.

The following figure shows the interface of this application.



Session name : Videoconference
Description :
Duration : [illegible]
Coordinator : dfuller

yes, I am ready

dfuller : are we ready to start?
lsuguerra : all member of the session are connected

all
dfuller
[illegible]

LOGIN : dfuller
NAME : David A. Fuller
EMAIL : dfuller@ing.puc.cl
ADDRESS : Av. Vicuña Mackenna 4860, Macul, Santiago - Chile
PHONE : [illegible]

Figure 5.1. *Electronic Meeting System application over TOP.*

For the implementation of this application we first create the *environment*. Then we define the *users* and two types of *roles*: coordinator and participants. For this application we wanted to store the data sent to the *broadcast*, so the *broadcast* box for this chat was defined with persistence option. We build a *session* object in which the coordinator determines the hour of beginning and ending of the meeting. We define three object types: "votes", "comments" and "microphone". On the "microphone" object type we use a *floor control*. We use a single microphone for the application. Only the user who has the microphone control can transmit audio and video.

Also in this application we have a voting system and a brainstorming system. When the coordinator decides it, he requests a voting session. He can set the voting button in each client, and the votes are sent to a voting *box*. The coordinator can also request a brainstorming session activating the button for this effect. The "ideas" type objects are sent to a brainstorming *box*. The brainstorming and voting boxes have defined an "orderly list" *view* type. Through this *view*, the coordinator can show to the meeting users the result of the voting or brainstorming sessions. These two options are carried out in an anonymous way. The system allows a single vote for person in each voting session.

6. Other Collaborative Applications

Other applications developed over TOP framework are: an asynchronous collaborative editor, a voting system, a brainstorming system a Web chat system and a collaborative Web navigator. For the voting and brainstorming system (similar to those used in the previous Electronic Meeting System) we define a *box*, an "orderly list" type *view*, one *object type* ("idea" and "vote") and the list of *users*, with rights to put and to see objects in the boxes.

For the Web chat we use a *broadcast*, a *session* and several *users*. In this application we use the *TopInterface* applet in order to communicate the application interface, build in JavaScript, with the TOP server. For the collaborative editor a *box* was used for store each document. We define two object types, text and graphics. Users were defined for each *box*, and for each one we create a reading *role* or a reading-writing *role*. The editor's users could only erase or modify objects if they were the owners of them. A *view* was used to keep the logical order of the document objects. The users could change the structure of that *view* to locate new objects in the corresponding place. This editor was very useful for collaborative writing of papers. At present we are working on an improved version of this editor. For the collaborative Web navigator, we have a coordinator user, who loads HTML pages on his browser. When the coordinator loads a new HTML page, a broadcast object send a notification to the other users in the session, which loads the same page the coordinator had in his browser. The coordinator role (*floorControl* object) can be passed through the session users.

Let us discuss a quick comparison among the GroupKit system [Roseman96] and the TOP framework. GroupKit is an extension to the Tcl/Tk language [Ousterhout94] that makes it easy to develop groupware applications to support real-time, distance-separated collaborative work between two or more people. GroupKit supports real-time distributed multi-point conferences between users. To begin the use of GroupKit, we need to create a session by the command `open.reg`. This instruction opens the *session manager* that shows a conference panel, as well as the connected users to them. We can build a similar application using the *session* object method that provides the session's list. This method shows all the open TOP sessions, and a user can join to one of these sessions if he has the corresponding access rights, defined by the corresponding *user*, *role* and *session* objects.

One of the main GroupKit commands is `gk_toAll`. The `gk_toAll` command is an example of a Remote Procedure Call (RPC) that GroupKit provides. This command executes a Tcl command on all processes in the session, including the local user. This command can be simulated in TOP by using a *broadcast* box in order to send commands or instructions to all the users connected in a *session*.

Other GroupKit instructions are `gk_toOthers` and `gk_toUsernum`. The first one executes a Tcl command on all remote processes in the session, but in the local user. The second one executes a Tcl command on only a single conference process, which may be one of the remote users or the local process. The *broadcast* TOP object has options to send objects to a subset of users in a session, so we can also simulate these GroupKit commands.

Other GroupKit facilities as the telepointers and multi-user scrollbars are more difficult to implement in TOP. These GroupKit facilities take advantage of the graphic Tk facilities. At this time TOP does not provide graphic facilities to support the human-computer interface. This interface should be built in some computer language.

When we want to create a collaborative application in the GroupKit platform, we need to write the entire code in the Tcl/tk languaje, and include the GroupKit commands. In the TOP framework, we just create the objects we need and build the user interface in the language we prefer, including JavaScript o Java for Web applications. In GroupKit we can not build Web applications. In TOP framework we have objects persistence, so we can build asynchronous collaborative applications. In GroupKit we can not create asynchronous applications. In addition, we can use TOP framework to build all the applications we build in GroupKit, but there are TOP applications that can not be build in GroupKit (Web applications, or asynchronous applications like the collaborative asynchronous editor). Also, the object-based architecture of the TOP framework makes the application construction process easier than the GroupKit and Tcl/Tk language.

7. Conclusions and Further Work

In this paper we present an object-based platform for the construction of collaborative applications, including Web collaborative applications. We discuss each object, including the instance variables and some of the main methods.

The *box* and *boxObject* objects represent a very flexible repository that allows creates and manage the applications group memory. The version option of the *boxes* allows creating data awareness mechanisms, by keeping object versions that could be shown to the users. The *boxObjectType* object allows defining new objects types to be stored in the boxes, as well as its visualization tags when these objects are shown as Web documents. The *view* object allows defining different views on the same objects stored in a *box*. The *floorControl* object allows defining floor-passing protocols over object types. The *user* and *role* objects allow to define users for the applications and to assign access rights over the objects of the *boxes* or *views*. The *session* object allows defining periods of collaborative work. It also allows extending single-group applications to multi-group applications. This object already allows re-using all the applications created on the framework, incorporating them as a session in new applications. Finally the *environment* object defines and relates the whole shared space of each application.

If we know the functionality of each object, we can design and build collaborative applications in a very fast way. The human-computer interface should be built in some computer language that allows creating sockets in order to communicate with the object server of the TOP platform. For Web applications we can use the *TopInterface* applet to communicate to the TOP server, using the JavaScript language to build the interface. The process of collaborative application construction over the TOP framework is easier and faster than the use of other platforms as GroupKit or Habanero.

At present, several applications have been built on the TOP framework. We are using some of those applications in a successful way. However, as line of future work, we are designing and building more complex applications and trying to improve and to include new services in the framework.

We are studying the possibility to incorporate an OODBMS in order to manage the objects in the boxes. We are also trying to improve the security and the concurrency control of the framework. We are also studying the possibility to include an HTTP server in the framework, in order to improve the construction of Web collaborative applications.

Acknowledgments

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

References

[Berners-Lee94] Berners-Lee T., Connolly D. *Hypertext Markup Language Specification - 2.0*. IETF HTML Working Group, RFC1866, 1994.

[Conklin92] Conklin J. *Capturing Organizational Memory*. Readings in Groupware and Computer-Supported Cooperative Work, San Mateo, CA, Morgan Kaufmann Publishers, 1993, pp. 561-565.

[Day97] Day M., Patterson J., Mitchel D. *The Notification Service Transfer Protocol (NSTP): Infrastructure for Synchronous Groupware.* Hyper Proceedings of the Sixth International World Wide Web Conference. Stanford, California, USA, 1997. Available at: http://www6.nttlabs.com/HyperNews/get/PAPER80.html.

[Dix96] Dix A. *Challenges and Perspectives for Cooperative Work on the Web*. Proceedings of the ERCIM Workshop on CSCW and the Web. Sankt Augustin, Germany, February, 1996.

[Dourish92] Dourish P. and Belloti V. *Awareness and Coordination in Shared Workspaces*. Proceedings of CSCW'92, pp. 107-114. Canada, 1992.

[Ellis91] Ellis C.A., Gibbs S J., Rein G.L. *Groupware Some Issues and Experiences*, Communications of the ACM, Vol. 34 not. 1, 1991, pp. 38-58.

[Flanagan97] Flanagan D., *Java in a Nutshell*. 2nd Edition, O'Reilly, 1997.

[Goldberg96] Goldberg M., Salari S. and Swoboda P. *World Wide Web–Course tool: An environment for building WWW-based courses*. Computer Networks and ISDN Systems, pp. 1219-1231. Proceedings of the 5th International World Wide Web Conference, Paris, France, 1996. Available at: http://www5conf.inria.fr/fich_html/papers/P29/Overview.html.

[Greenberg96] 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 24-27, 1996.

[Guerrero97] Guerrero L., Fuller D. *CLASS: A Computer Platform for the Development of Education's Collaborative Applications.* Proceedings of CRIWG'97. Madrid, Spain, 1997.

[Ousterhout94] Ousterhout, J. *Tcl and the Tk Toolkit*. Addison-Wesley, USA, 1994.

[Patterson96] Patterson J., Day M. and Kucan J. *Notification Servers for Synchronous Groupware.* Proceedings of CSCW'96, pp 122-129. Boston, USA, 1996.

[Pieper97] Pieper M. and Hermsdorf D. *BSCW for Disabled Teleworkers: Usability Evaluation and Interface Adaptation of an internet-based Cooperation Environment.* Hyper Proceedings of the Sixth International World Wide Web Conference. Stanford, California, USA, 1997. Available at: http://www6.nttlabs.com/HyperNews/get/PAPER161.html.

[Roseman96] Roseman M. and Greenberg S. *Building Real Time Groupware with GroupKit, A Groupware Toolkit*. ACM Transactions on Computer Human Interaction, 3(1), p66-106, March, 1996.

[vanWelie96] vanWelie M. and Eliëns A. *Chatting on the Web.* Proceedings of the ERCIM Workshop on CSCW and the Web. Sankt Augustin, Germany, February, 1996.

[Wooldridge97] Wooldridge A., Morgan M., Reynolds M.C. and Honeycutt J. *Using JavaSript*. Second Edition. QUE. 1997.

URL References

[BSCW] Basic Support for Cooperative Work. http://bscw.gmd.de.

[Groupkit] Groupkit. http://www.cpsc.ucalgary.ca/projects/grouplab/projects/groupkit.

[Habanero] Habanero. http://www.ncsa.uiuc.edu/SDG/Software/Habanero.