CoLab: A Flexible *Collaborative Web Browsing* Tool[⊕]

Guillermo de Jesús Hoyos-Rivera[§]
LAAS – CNRS[¤]
ghoyos@laas.fr

Roberta Lima-Gomes[+]
LAAS – CNRS[¤]
rgomes@laas.fr

Jean-Pierre Courtiat
LAAS – CNRS[¤]
courtiat@laas.fr

Abstract

Collaborative Web Browsing aims at extending currently available Web browsing capabilities to allow several users getting their browsing activity synchronized. This is a new and promising research area whose possibilities have not yet been exhaustively covered in either, research or commercial solutions.

From our point of view a Collaborative Web Browsing system should provide all the necessary facilities to allow users to get synchronized and desynchronized in a flexible way. This synchronization should rely on a simple authorization protocol through which users can invite other users to create and release browsing synchronization relations. Additionally, special synchronization operators are also proposed with the purpose of overriding the authorization protocol. These operators are intended to be used by users having special privileges.

In this paper we present our proposal for modeling and implementing of a Collaborative Web Browsing system called CoLab [1], which has been developed using pure Java™ technology; we present the main features of our current development prototype: CoLab 2.0 which relies on the previously stated characteristics.

Keywords: Collaborative, Web, Browsing, Protocol, Synchronization.

1. Introduction

Collaborative Web Browsing can be seen as an extension of traditional Web browsing. The latter consists essentially of users accessing resources that are available in Web servers connected to a network (*Internet* or *Intranet*). However at the present time when users browse the Web they are completely isolated since they have no way of sharing online their browsing activities with other users. *Collaborative Web Browsing* overcomes this problem allowing users to "browse together".

There are several ways from which this issue can be tackled. The one that we have chosen consists in offering users the capability of easily creating and releasing synchronization relations among them as they wish. We understand a synchronization relation as the fact of binding the Web browsing of one user, called a synchronous user, to that of another one, called an asynchronous user. In this way the browser of the synchronous user will automatically retrieve and present the same Web pages sequence than the one requested by the asynchronous user.

This way of working opens new possibilities in collaborative work since it breaks the currently existing isolation of users associated with Web browsing activities. As a result, collaboration relations can emerge dynamically as users browse the Web, discover new material, and share it online with other users. In this way a new collaborative dimension is added to the Web browsing paradigm.

The paper is organized in 6 main sections. In section 2 we present an overview of the *Collaborative Web Browsing* concept and discuss other related proposals, justifying then our approach. In section 3 we present the architecture of our system and explain its main characteristics giving an overview of the whole specification. In section 4 we present the notions behind the users' synchronization model, and we explain its main characteristics. In section 5 we present the current state of implementation of our platform and we explain its operation. Finally in section 6 we draw some conclusions and discuss future work.

[¤] Laboratoire d'Analyse et d'Architecture des Systèmes. 7, Av. du Colonel Roche, 31077, Toulouse, France.
[⊕] This research is supported by the European IST project *Lab@Future*.
[§] Researcher of the Universidad Veracruzana, México, financially supported by scholarship 70360 from CONACyT and by PROMEP, México.
[+] Researcher financially supported by a scholarship from CNPq, Brazil.

2. Overview of the *Collaborative Web Browsing* Field

The *Collaborative Web Browsing* area has taken a growing relevance in the last years due mainly to its potential as a way of creating new ways of cooperation among Web users. Accordingly several people in both, research and commercial fields have developed different models and proposals in this area. We have made a detailed state of the art in this research area, leading us to the following references which are the most representative:

- *Ariadne* [2] is a system that allows collaboratively indexing in bibliographical databases, that means that while several users browse independently on bibliographical resources they qualify them into categories, and from these qualifications Ariadne creates an indexation of them.
- *CoBrow* [3] is a system that creates logical neighborhoods related to the contents of the document currently being browsed by several users; this makes it possible to ensure awareness among users browsing similar documents, facilitating therefore their interaction.
- *E-CoBrowse* [4] is a system that uses Chatpointer (tele-pointers with chat-style conference capabilities) which allow the users to communicate while they are browsing.
- *Let's Browse* [5] is an experiment in building an agent to assist a group of people in browsing, by suggesting new material likely to be of common interest. It is built as an extension to the Web browsing agent Letizia, which does a real-time, incremental breadth first search around the user's current page, and filters candidate pages through profiles learned from observing the users' browsing activity.
- *PROOF* [6] is a system that provides awareness of the browsing activities executed by users registered in a session, as well as browsing synchronization features implemented in a centralized way.
- *WebSplitter* [7] is a system that offers adapted versions, when available, of the retrieved resources depending on the technical constraints of the device used for accessing the Web (PC, PDA, etc.).
- *WikiWikiWeb* [8] is a system that contains a collection of Web pages, and that allows its users to create, delete and modify them from anywhere and at anytime. Users interact via these pages.
- *Commercial tools - NetDive* [9], *PlaceWare* [10], *WebCT* [11], *WebEx* [12] are integrated collaborative environments that allow users to access a session and communicate and interact by the use of several available communication tools. All of them include a synchronized browsing facility. A *Floor Control* system is commonly used to control the exclusive access to shared resources.

All these proposals offer alternatives for collaboratively browsing the Web in some degree. The main difference of our approach with respect to the previous ones is that we focus on the dynamic creation and release of synchronization relations among the users. The creation of synchronization relations among users, as we said in the introduction, leads to binding the Web browsing of a user to that of another user. This way of working can be seen as an extension to the classical *Floor Control*, where in the presence of a synchronization relation the user who has the floor is the asynchronous one.

The creation of a synchronization relation implies using an authorization protocol that takes the form of an invitation, which may be accepted or refused. If accepted, the synchronization relation creation succeeds and the user that gets synchronized looses his floor; otherwise the synchronization state of the concerned users keeps unchanged.

Additionally to this basic synchronization protocol we consider also to include a special synchronization mode through which the authorization protocol can be overridden. This is based on the notion of privileges, in such a way that privileged users can unconditionally create synchronization relations.

3. The Architecture of the *Collaborative Web Browsing* System

In this section we will present the architecture of our Collaborative Web Browsing system. In order to explain it we present the architecture diagram in Figure 1.

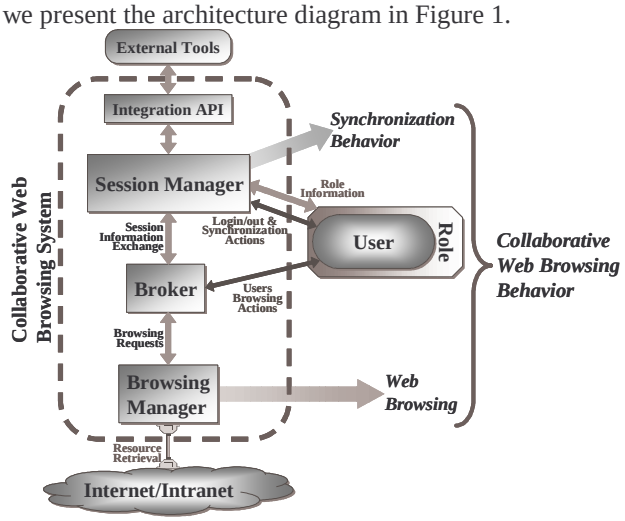

Figure 1. Collaborative Web Browsing architecture

In this figure we can see the main components of our model. These are the *Session Manager*, the *Broker*, and the *Browsing Manager*.

The *Session Manager* is in charge of managing the *Collaborative Web Browsing* session itself. It is responsible for keeping track of the connected users, as well as of the existing synchronization relations. It is also responsible for the eventual integration of the system via the *Integration API* presented at the top in this figure, with other systems or additional modules.

The *Browsing Manager* is in charge of all the tasks related to the retrieval of resources from the network. This is not directly accessed by the users, but though the *Broker*, which acts as an intermediary between these two. The reason is that browsing requests are not to be systematically satisfied, but they depend on certain conditions verified at the level of the *Session Manager*. In this way, when a browsing request arrives, the *Broker* asks the *Session Manager* to verify whether or not a given request is to be satisfied.

Aiming to give a more complete view of how all the components of our model fit with each other, the *UML Class Diagram* is presented in Figure 2.

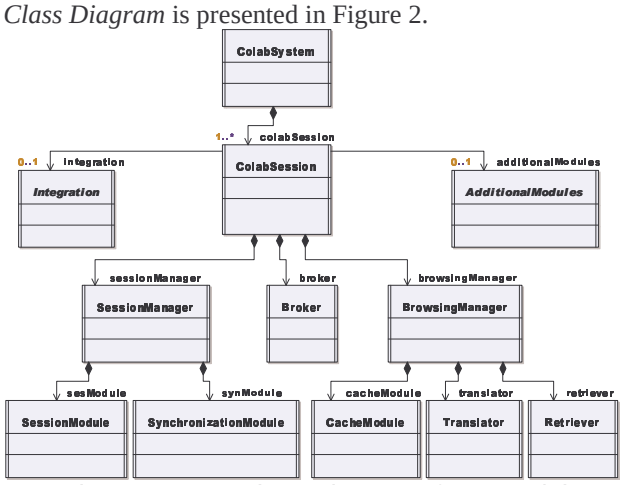

Figure 2. UML Class Diagram of our model

At the top of this diagram we can see the representation of the *Collaborative Web Browsing* system itself. This is capable of managing concurrently one or more *Collaborative Web Browsing* sessions. Each of these sessions has its own operation parameters defined (e.g. initial URL, roles, privileges), and workspaces of the sessions are disjoint. Each session is constituted of the main components previously illustrated in the Figure 1; here we go one step beyond in the model's details since we can see that the *Session Manager* and the *Browsing Manager* are composed of sub-modules.

The *Session Manager* includes two main components: the *Synchronization Module*, which is in charge of treating all the synchronization actions, and guaranteeing the overall consistency of the synchronization state (this subject will be analyzed later in more detail), whereas the *Session Module* implements the core functions of the *Session Manager* itself.

The *Browsing Manager* includes three main components: the *Cache Module*, the *Retriever*, and the *Translator*. All these components interact among them in order to satisfy incoming browsing requests.

The *Cache Module* corresponds to the implementation of a cache system which allows storing locally a copy of every retrieved resource, in such a way that it is not necessary to retrieve the original resource if this is already stored locally, and that the local copy has not expired with respect to the original version. This module, as we will see later, is systematically called when the browsing request comes from a synchronous user.

The *Retriever* directly responsible of retrieving every requested resource. It can be retrieved directly from the network or from the *Cache Module*. In the case of a resource retrieved from a Web server, the *Retriever* sends it to the *Translator* in order to be treated before sending the response to the user's browser. In this case, once the resource has been translated, the result is sent back to the *Retriever* and also to the *Cache Module* to be stored for subsequent use.

The *Translator* is responsible of modifying every retrieved HTML resource. This is necessary due to the fact that HTML resources must be modified before sending them to the users' browsers in order to allow our system to track the users' browsing actions.

Finally we can see the two optional components which constitute the *Integration API* identified in the diagram as *Integration*, and *Additional Modules*. The *Integration* component takes in charge all the necessary tasks to allow our *Collaborative Web Browsing* environment to be integrated to other systems, for example external communication tools, in order to enrich the Collaborative Web Browsing behavior. On the other hand the *Additional Modules* component helps to the integration of new capabilities to the Collaborative Web Browsing behavior (for instance, an *Access Control Module* for restricting the access to some Web resources, an *Adaptation Module* for adapting presentation of Web resources, etc.).

3.1. Operation behavior of *CoLab*

Aiming to graphically illustrate the operation behavior of our proposal, and how the different components of our model interact, we present in Figure 3 the case of a typical browsing action executed by a user, and the resulting synchronization of the browsing action to other user.

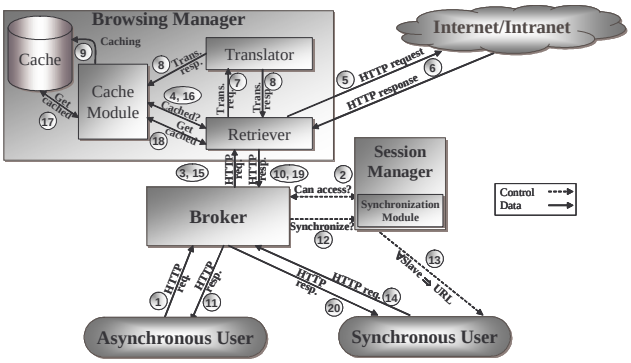

Figure 3. Synchronization of the browsing actions

The process can be logically separated in two activities. The first one deals with the retrieval of a resource for the asynchronous user, and the second, as result of the synchronization of the browsing action, the retrieval of that resource for the synchronous user.

The first step consists in the request of a resource expressed by the asynchronous user (1), which is treated directly by the *Broker*. Next the *Broker* contacts the *Session Manager* to ask it whether the user can access the requested resource (2). If so the *Broker* sends the request to the *Retriever* (3), who asks the *Cache Module* if that resource can be found in the local cache (4). Let's assume that this is not the case, so the resource is retrieved directly from the remote Web server (5-6), and if it is identified as a HTML one, it is sent to the *Translator* in order to be modified (7). Once the resource has been translated, it is sent back to the *Retriever* (8), and also to the *Cache Module* in order to store it in the local cache (8-9). At this moment in the process the *Retriever* sends the resource back to the *Broker* (10), and this last sends it to the user's browser which has made the request (11).

Once the previous steps have been completed, the *Broker* asks the *Session Manager* to synchronize this browsing action for all the users that are currently synchronized with the user who has just browsed (12). Face to this request, the *Session Manager* sends this command to every single browser of each synchronous user (13). Each browser will then make its own request for the indicated resource (14), which will be sent again to the *Broker*. The *Broker* asks the retrieval of the resource to the *Retriever* (15), which asks the *Cache Module* to verify whether it is cached or not (16). As the resource has been already stored in the local cache, and this browsing action is the product of a synchronization, it is retrieved directly from there (17) and sent back to the *Retriever* (18), which sends it back to the *Broker* (19), for finally satisfying the user's request (20).

The process depicted in the Figure 3 can be viewed also as a Sequence Diagram, which is presented in Figure 4. Here the steps marked as (A) and (B) represent the login operations of two users, and the step marked as (C) represents the operation through which both users get synchronized (user "memo" gets synchronized with user "beta").

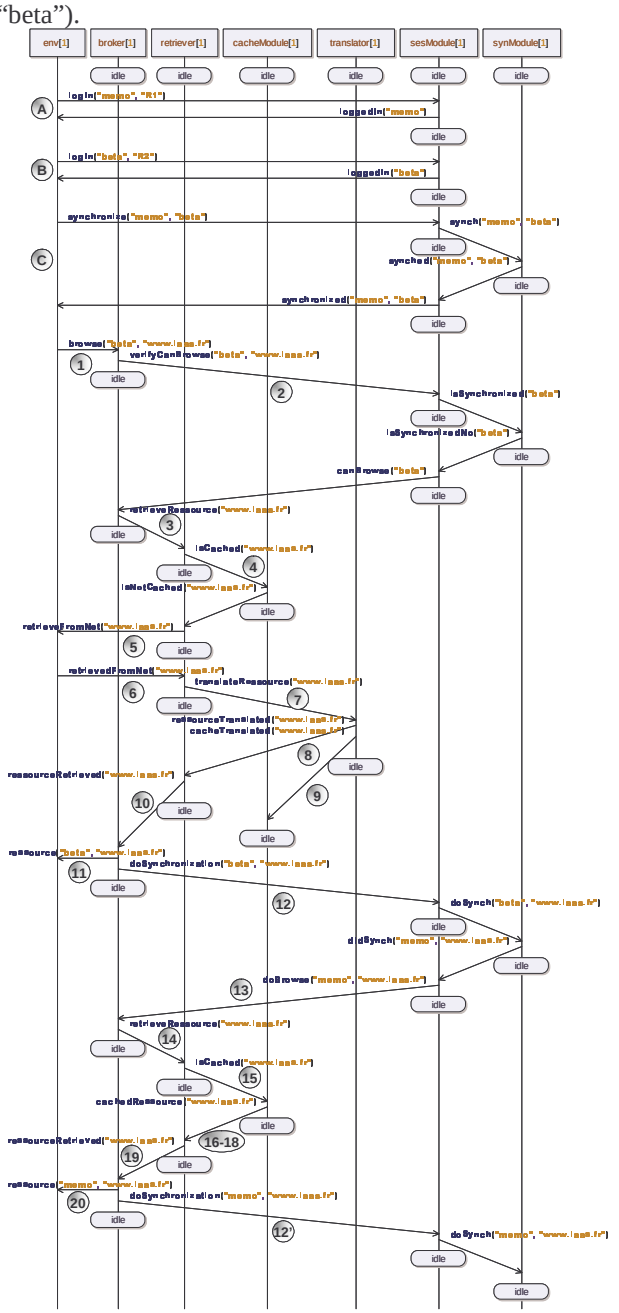

Figure 4. Collaborative Web Browsing sequence diagram

4. The Synchronization Model

The core element of our *Collaborative Web Browsing* proposal is the synchronization model. For the representation of the synchronization relations between the users we have decided to use a tree structure which is called *SDT* (*Synchronization Dependency Tree*). In a *SDT*

the nodes denote the connected users, and the arcs denote the synchronization relations currently existing among them. For example for any pair of nodes *A* and *B*, if node *A* is the father of node *B*, then the browsing actions of user *B* are subordinated to those of user *A*. The root node of each *SDT* represents an asynchronous user, that means, a user who is free to browse. The other users belonging to the same *SDT* are called synchronous users. That means that their browsing actions are synchronized with the browsing actions of their root node. In terms of *Floor Control*, we can say that the root node of any *SDT* corresponds to the user who actually has the floor. We have chosen the tree structure given that a single user can be synchronized with only one user at a time, however a single user can have any number of users synchronized with him. The basic notion of *SDT* is presented in Figure 5.

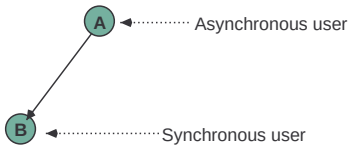

Figure 5. Basic notion of SDT

In our representation the direction of an arrow denotes where the propagation of the synchronization goes to. In Figure 5, we have two connected users, *A* and *B*, and user *B* is currently synchronized with user *A*.

At any moment in a given session there may be as many *SDT*s as asynchronous users. Whenever two users decide to get synchronized their *SDT*s merge becoming a single *SDT*. The maximum number of possibly existing *SDT*s in a session is the number of connected users when all of them are working asynchronously. The minimum number of *SDT*s in a session is one, and is the case when all the users belong to the same *SDT*. The number of *SDT*s actually existing in a session is called the *SDT Cardinality*, and it is denoted by |*SDT*|.

In order to illustrate these notions we present in the Figure 6 some possible configuration scenarios of the *SDT*s in a session.

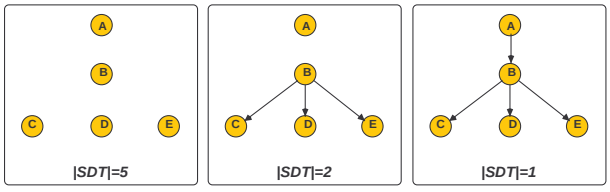

Figure 6. Scenarios of configuration of SDTs in a Collaborative Browsing session

The first scenario represents the case when every user is working asynchronously (i.e. each one has his own floor), so that |*SDT*|=5. As users create or destroy synchronization relations, the number of existing *SDT*s changes. In the second scenario we can see that users *C*, *D* and *E* have got synchronized with user *B* (user *B* has the floor for users *C*, *D* and *E*), which means that whenever user *B* executes a browsing action, users *C*, *D* and *E* will be forced to execute the same action. In this case |*SDT*|=2. In the case of the third scenario we can see that a single *SDT* may in fact have several levels. In this case user *B*, who in scenario 2 was asynchronous, has decided (or has been invited) to get synchronized with user *A*, (user *A* has the floor for all the users of the session) so whenever this last executes a browsing action, all the other users will execute the same browsing action. In this last case |*SDT*|=1.

4.1. The synchronization operators

In its basic operation mode, there are two operators that allow the creation of synchronization relations among users: *I_Follow_You* and *You_Follow_Me*. The first one expresses the intention of a given user to get synchronized with another user. The second one expresses an invitation for another user to get synchronized. Given that a single *SDT* node may have several children, the *You_Follow_Me* operator has the characteristic that it can be applied to a single user as well as to a set of users.

Whenever any of these two operators is applied, the authorization protocol is started in such a way that the user who the invitation was sent to is asked whether he wants or not to accept it. The synchronization relation will be created only if the user agrees; otherwise no modification will be made to the existing *SDT*s. Synchronization relations are released by using the operator *I_Leave*, which is unconditional, that means that any of the involved users can apply it, and it will always succeed.

In order to illustrate the general behavior of the synchronization process, in Figure 7 we use extended state machine-style notation.

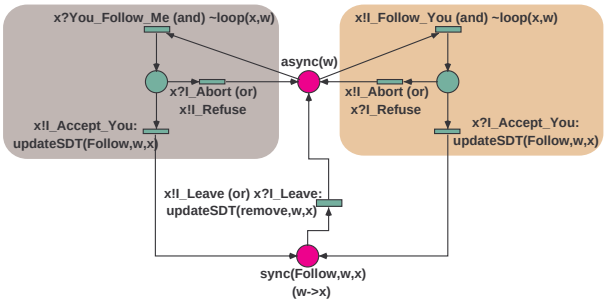

Figure 7. Behavior of the users' synchronization process from the point of view of user W

In this figure the two main states in which user W can be are *async*: the user is working asynchronously, and *sync()*: the user is currently synchronized with another user. When the user is in the *async* state he can either, express the intention of getting synchronized with another

user (right colored area), or receive an invitation from another user to get synchronized with him (left colored area). In any of these two cases the preconditions to be able to apply the operator are that: i). user *W* is asynchronous, and ii). the tree structure is respected (e.g. no loops are created). Then the system passes to an intermediary state where the invitation is expressed, and keeps waiting for an answer to the request: an acceptance, a refusal or an abort. If operation does not succeed, user *W* gets back to work in asynchronous mode, otherwise the synchronization relation is created, and the involved *SDT*s merged in a single *SDT*.

In order to validate our model we have implemented the behavior of the operators *I_Follow_You*, *You_Follow_Me* and *I_Leave*, as well as the *Browse* operation, using Petri nets using the TINA tool developed at LAAS-CNRS [13], and we have demonstrated that our model is consistent. These demonstrations are beyond the scope of this paper and are not presented here.

As previously said, we propose two additional synchronization operators which are intended to allow privileged users to create synchronization relations without requiring any authorization. These are *I_Spy_You* and *You_Join_Me*. The first one allows a given user to get unconditionally synchronized with another user. In this case the only user in the *Collaborative Web Browsing* session who is aware of the creation of that synchronization relation is the one who applied it. The second one allows a given user to force another user, or a set of users, to get synchronized with him. In both cases the only user who can release the synchronization relation is the one who created it.

5. *CoLab*'s current implementation

At the present time we have developed *CoLab* version 2.0, which implements almost all the concepts presented in the previous sections. It has been implemented on a *PC* with the *Linux RedHat 7.2 OS*. The software choice for developing *CoLab* consists of the *Java™ 2 SDK Standard Edition* release 1.3.1_05, *Jakarta™ Tomcat* release 3.3.1a for the *Servlets/JSP* technology, and *JSDT* release 2.0 for the *CoLab*'s internal communication facilities. On the browser side the only technical requirement is that it supports Java™ Applets, and that it can implement the *Automatic Proxy Configuration* (*PAC*) facility. In the next sub-sections we will present the main features of *CoLab*'s implementation and describe its main characteristics and operation behavior.

5.1. *CoLab*'s implementation architecture

In terms of implementation the Figure 8 illustrates the *Colab*'s implementation architecture, introducing its main components: the *CoLab Server*, the *Integrated Applications Servers*, the *Clients*, and the underlying *Communication Network*.

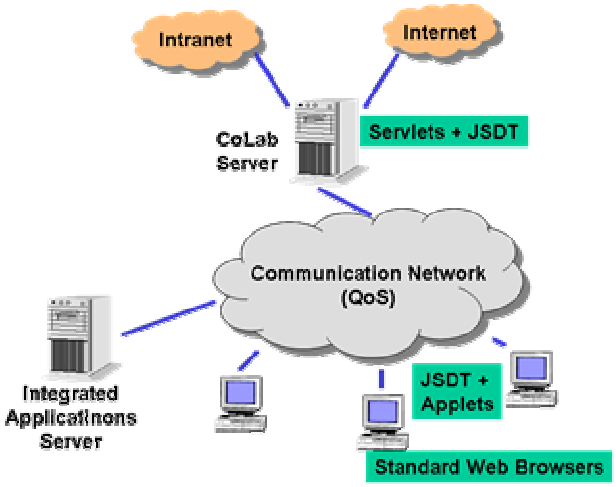

Figure 8: *CoLab*'s implementation architecture

At the center of this diagram there is the Communication Network, through which all the components of the system communicate, and from which the requested resources are retrieved. There is no a-priori assumption about the underlying communication network, with the exception that it should offer enough bandwidth to ensure an adequate level of *QoS*. End to end communication is based on *TCP/IP* (*HTTP*/*TCP*/*IP* for the access to the *Web* pages, *RTP*/*UDP*/*IP* for the continuous streams). Current experimentations are carried out on a 100 Mbps switched *Ethernet* local area network, however future developments will address specifically the *QoS* issue within the context of *WAN/LAN* communication network.

One potential problem of the current *CoLab* architecture, as presented in this section, is related to its scalability, in particular at the level of the *CoLab Server*, which plays a fundamental role. In the current implementation, we consider a centralized architecture. Further study dealing with the distribution of the *CoLab Server* to balance the workload among distributed servers will be initiated after a careful analysis of the performance and scalability issues of the current architecture.

5.2. *CoLab* current operational implementation

CoLab platform is oriented to non-expert users. As a consequence, we have decided that it must be simple and easy-to-use. The first step in order to use CoLab is to configure the sessions that will be available.

Configuration is done via an XML file which contains the specification of the default initial page, the available roles, and the eventual existing privileges that can be associated to each pair of them. An example of configuration file is presented in Figure 9.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<colab_session url="http://java.sun.com">
   <role_definition>
      <role role_name="Teacher"
         role_password="teachpass"/>
      <role role_name="Assistant"
         role_password="assistpass"/>
      <role role_name="Student"
         role_password="studpass"/>
   </role_definition>
   <role_privileges>
      <can_spy from="Teacher" to="Student"/>
      <can_force from="Teacher" to="Student"/>
      <can_spy from="Assistant" to="Student"/>
   </role_privileges>
</colab_session>
```

Figure 9. Typical session configuration file

In this file, for example, we can see that the home URL for the session is *http://www.laas.fr*, and that there are three available roles: *Teacher*, *Assistant* and *Student*. Concerning the privileges, we can see that users having assumed the role *Teacher* can apply the *I_Spy_You* and the *You_Join_Me* operators on users having assumed the role *Student*, and that users having assumed the role *Assistant* can apply the *I_Spy_You* operator on users having assumed the role *Student*. Once the sessions have been configured, the *CoLab* server can be started.

On the users' side, in order to be able to access the available *CoLab* sessions, users must configure in their preferred browser the *Proxy Automatic Configuration* facility in order to point the place where the configuration file can be found. This file contains a specification in JavaScript™ that allows redirecting browsing request depending on certain criteria. For example, in Netscape this configuration can be made in the Preferences menu option, as indicated in the Figure 10.

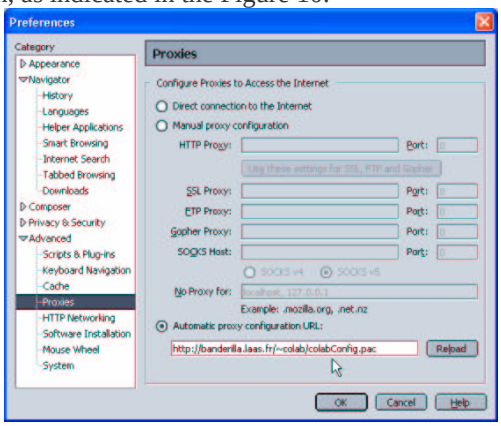

Figure 10. PAC file configuration

When accessing CoLab, the first step is to choose a session from those available. Once a session has been chosen, the roles available for that session are presented in the login screen. Then the user must choose the role he wants to assume, type the password associated to this role, and choose a username, through which he will be identified. The login screen is presented in the Figure 11.

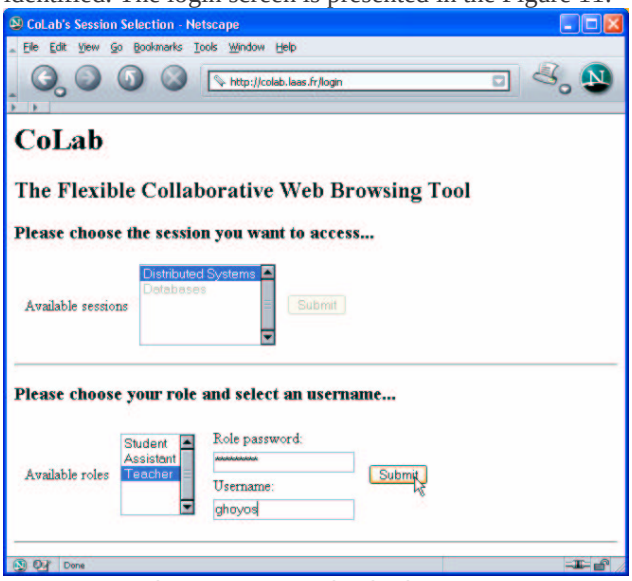

Figure 11. *CoLab*'s login page

Once the user has been authenticated a new browser window opens, the one where *Collaborative Web Browsing* activities will take place. The other browser window can be used for browsing the Web outside the *CoLab* session.

The *CoLab* session window has two frames: one of them contains the *Control Frame* where browsing and synchronization controls are located, and the other contains the *Browsing Frame*, where the browsed pages will be presented. A screen capture of the browsing and synchronization controls is presented in the Figure 12.

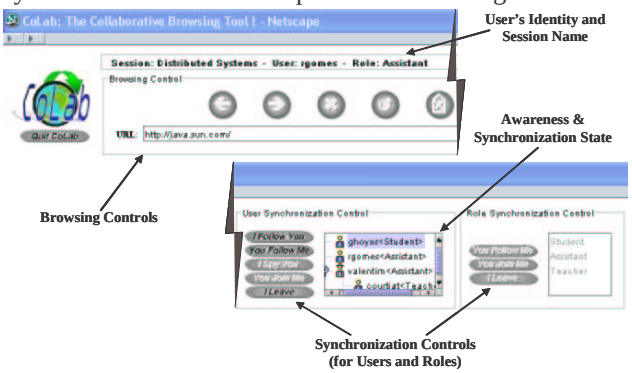

Figure 12. The browsing and synchronization controls frame

The *Control Frame* contains a *Graphical User Interface* (*GUI*) providing all the necessary components to make *Collaborative Web Browsing* possible. On the left side (presented at the top of Figure 12) are the *Browsing Controls*, which are equivalent to those of a typical

browser. The only special characteristic is that the "Home" button will load the default initial web page for the session, as it was defined in the session configuration file. On the right side are the *Awareness and Synchronization Controls*. There are presented: i). the user and current synchronization state awareness, and ii). the available synchronization controls associated with the synchronization operators explained in the previous sections. Here we can see which users are currently present in the session, and which are the existing synchronization relations. For example, in the image presented in this last figure we can see that there are four users currently logged in the session: *ghoyos*, *rgomes*, *valentim* and *courtiat* and we can also see that the user *courtiat* is currently synchronized with user *valentim*. Concerning the synchronization controls we can see that they are divided in two sections: *User Synchronization* and *Role Synchronization*. The first one contains the buttons representing the synchronization operators that can be applied to single users, and the second has only the buttons corresponding to the synchronization operators that can be applied to the roles.

6. Conclusions and future work

In this paper, we have defined a general-purpose *Collaborative Web Browsing* system, which provides a new paradigm since it offers to the users the possibility of easily creating and releasing browsing synchronization relations among them. We think that this orientation gives the users a lot of flexibility for establishing collaboration relations while they are browsing, creating in this way an environment where collaboration among users is greatly facilitated by allowing them to synchronize their browsing activities.

The current operational implementation of this system, supporting only a subset of the whole functionality, has been developed and is operational on a switched *LAN*. Given that the only information that is exchanged between the connected clients and the server consists only of short messages associated with the synchronization protocol, and URLs to achieve the synchronization of the browsing activities, and that the synchronized resources are directly retrieved from the local cache system, we can affirm that there is practically no overload associated to the operation of the system itself.

For the next future we will keep working in the implementation of all the features of our model, as well as identifying new opportunity areas where we can improve CoLab's capabilities, as the possibility of adding annotations to the browsed resources in order to facilitate the information exchange among the users.

We will also start working on the implementation of the *Integration API*, identifying the possible requirements to be satisfied in order to get *CoLab* integrated with other collaborative systems and tools.

Another subject on which we will start working soon is the implementation of the distributed version of our platform in order to avoid any performance bottlenecks in presence of heavy workload.

7. References

[1] Hoyos-Rivera, G.J.; Lima-Gomes, R. & Courtiat, J.P. "A Flexible Architecture for Collaborative Browsing", *11th IEEE WetICE, Workshop on Web-Based Infrastructures and Coordination Architectures for Collaborative Enterprises*, 2002, Carnegie-Mellon University, Pittsburgh, PA, USA

[2] http://www.comp.lancs.ac.uk/computing/research/cseg/projects/ariadne/

[3] Sidler, G.; Scott, A. & Wolf, H. "Collaborative Browsing in the World Wide Web",*JENC8*, 1998, pp. 221-1 – 221-8.

[4] Chong, S.T. & Sakauchi, M. "E-CoBROWSE: co-Navigating the Web with Chat-pointers and Add-ins – Problems and Promises". *IASTED ICPDS, Collaborative Technologies Symposium*, ACM, Las Vegas, Nevada, USA, Nov. 2000.

[5] Lieberman, H.; Van Dyke, N.W. & Vivacqua, A.S. "Let's Browse: A Collaborative Web Browsing Agent", *IUI'99*, ACM, Redondo Beach, CA, USA, 1999.

[6] Cabri, G.; Leonardi, L. & Zambonelli, F. "A Proxy-based Framework to Support Synchronous Cooperation on the Web", Software – Practice and Experience 29(14), John Wiley & Sons, Ltd., 1999

[7] Han, R.; Perret, V. & Naghshineh, M. "WebSplitter: A Unified XML Framework for Multi-Device Collaborative Web Browsing". *CSCW'00*, IEEE, Philadelphia, PA, USA, Dec. 2000.

[8] http://c2.com/cgi/wiki

[9] http://www.netdive.com/

[10] http://main.placeware.com/

[11] http://www.webct.com/

[12] http://www.webex.com/

[13] http://www.laas.fr/tina