CoLab: A New Paradigm and Tool for Collaboratively Browsing the Web

Guillermo de Jesús Hoyos-Rivera, Roberta Lima Gomes, Roberto Willrich, and Jean-Pierre Courtiat

***Abstract*—Widespread adoption of Web technologies, particularly in professional and educational areas, has motivated new research efforts with the objective of designing new interaction mechanisms based on Web technologies. Within this framework, collaborative Web browsing (cobrowsing) aims at extending currently available Web browsing capabilities in order to allow several users to "browse together" on the Web. Such a browsing paradigm can have many useful applications, for instance, in e-learning, for collaboratively searching and retrieving documents, and for online assistance (helpdesk). A cobrowsing system should provide all the facilities required for allowing users to establish and release, in a very simple and flexible way, browsing synchronization relations as well as interactions with continuous media presentations embedded within Web pages. This paper presents the design, modeling, and implementation of the cobrowsing system called CoLab. CoLab provides all the functionalities required for allowing users to collaboratively browse the Web, and a first experimental version of the tool has been implemented and is fully operational.**

***Index Terms*—Collaborative Web browsing, continuous media, synchronization.**

I. Introduction

THE World Wide Web (WWW) is a large distributed collection of documents connected by hypertext links. Web browsers are the basic tools for accessing and displaying these documents. Although this collection of documents can be concurrently accessed by several users, Web browsers are basically single-user tools. Accordingly, users are isolated when browsing the Web since they have no way of sharing online their browsing activities with other users. A great effort must be made to allow a group of users to share their browsing activity (for instance, the pages they have visited).

"Collaborative Web browsing" overcomes this problem by allowing users to "browse together." In this paper, we consider a cobrowsing system as a tool for allowing users to browse Web pages together in cobrowsing sessions while establishing/releasing browsing synchronization relations as they wish. This way to proceed 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 dynamically emerge as users browse the Web, discover new material, and share it online with other users, adding in this way a new dimension to the Web browsing paradigm.

Several application fields can take advantage of this new collaborative Web browsing paradigm.

- In an e-learning environment, cobrowsing can be used effectively in many situations to increase productivity. For instance, teachers can present Web-based lectures to multiple students; teachers and/or students can collectively explore information on the Web (or in a digital library); and to answer student's questions, the teacher can cobrowse Web-based educational materials with a single student as well as with a group of students.
- Cobrowsing permits implementing collaboration and cooperation in digital information environments where information seekers can interact with other users and ask for help, and work with information in a group [1].
- In commercial environments, there are several applications for cobrowsing [2]: to provide assistance when a user has a question while browsing a company's website, to show to clients new items and promotions, to help filling Web-based forms, etc.
- Cobrowsing allows collaborative Web browsing support of materials during teleconferences, which would represent a helpful support for presenters to synchronously show information to attendees.

There are several requirements that a cobrowsing solution must meet. We believe that one of the most important is to provide flexible capabilities for organizing cobrowsing sessions. Such an organization defines which users are authorized to follow a link and when and which user(s) should automatically retrieve a given resource.

Most current cobrowsing solutions adopt two types of organization for a cobrowsing session: unmanaged or centralized. In an unmanaged organization, any member can follow a link while the other members will follow it automatically. This way of working could turn the cobrowsing session uncontrollable for groups of more than three users. Conversely, in a centralized organization, each session has a leader who controls the browsing actions. This organization type is only suitable for cobrowsing sessions where the browsing actions of the leader must be

Manuscript received October 3, 2005; revised May 1, 2006 and July 17, 2006. This work was supported in part by the European IST project Lab@Future. The work of G. J. Hoyos-Rivera was supported in part by the CONACyT under Grant 70360 and in part by a contract from the LAAS-CNRS in France. The work of R. Gomes was supported by a grant from the CNPq, Brazil. The work of R. Willrich was a cooperation between LAAS-CNRS and UFSC funded by a grant from CAPES-COFECUB. This paper was recommended by Guest Editor G. Cabri.

G. J. Hoyos-Rivera is with the Universidad Veracruzana, Xalapa 91000, México (e-mail: ghoyos@uv.mx; ghoyosr@gmail.com).

R. L. Gomes is with the Federal University of Espirito Santo, 29060-900 Vitoria, ES, Brazil (e-mail: rgomes@inf.ufes.br).

R. Willrich is with the Computer Science Department, Federal University of Santa Catarina, 88040-900 Florianopolis, SC, Brazil (e-mail: willrich@inf.ufsc.br).

J.-P. Courtiat is with Laboratoire d'Analyse et d'Architecture des Systemes-Centre National de la Recherche Scientifique (LAAS-CNRS), 31077 Toulouse Cedex, France (e-mail: courtiat@laas.fr).

Digital Object Identifier 10.1109/TSMCA.2006.883173

followed by all the other session members (for instance, when a teacher must present Web-based lectures to several students).

An alternative proposal for organization of cobrowsing sessions allowing dynamic organization of session is presented in this paper. Here, session members can dynamically reorganize the cobrowsing session in workgroups. A workgroup is composed by one or more session members whose browsing activities are synchronized. Workgroups can be dynamically created and modified. Therefore, beyond the centralized organization (where all the session members compose one workgroup), our solution allows creating a permanently or temporally decentralized organization. Workgroups can be temporally decentralized, and later, some of them can be merged together. This approach allows implementing the concept of "divide to conquer" [15], which is very important for several applications.

In this paper, we propose a cobrowsing system called CoLab [3]. This cobrowsing system is based on a simple and powerful synchronization model supporting a dynamic organization of a cobrowsing session. The proposed model offers a simple mechanism allowing session members to create and release synchronization relations among them.

This paper is organized as follows. In Section II, we present the basic requirements of cobrowsing systems and identify some currently implemented solutions and cobrowsing tools. In Section III, we present the proposed cobrowsing synchronization model and explain its main characteristics. In Section IV, we describe the architecture of our system and detail its main characteristics, giving an overview of the whole specification. The operational behavior of CoLab is presented in Section V. In Section VI, we present the current implementation of our platform, explaining its operation and presenting some performance tests as well as an experiment with real users. Finally, in Section VII, we draw some conclusions and discuss future work.

II. Cobrowsing Requirements and Solutions

A. Cobrowsing Requirements

In this section, we outline some basic requirements for a system aiming at providing generic cobrowsing capabilities.

1) Session Management: A cobrowsing system must allow the dynamic creation of cooperative multigroup multiuser cobrowsing sessions and provide facilities for the management of groups of users in each session. The system should provide user authentication, for controlling user access to a given session, and authorization, for defining user capabilities, including what kind of synchronization relations he is authorized to establish.

2) Cobrowsing Organization: Cobrowsing solutions must emulate social rules and social protocols that allow controlling collaborative work organization. For this reason, some kind of floor control must be implemented for each cobrowsing session. A cobrowsing solution should support different types of organization in order to be adapted to several types of cobrowsing application scenarios. Moreover, as in any people's organization, the function of a session member can change, so the system must be capable of coping with this kind of situations.

3) Web Browsing Synchronization: In a cooperative session, any browsing action done by some user X must be replicated as fast as possible in the browsers of all the users synchronized with user X. Accordingly, each browser must implement a shared workspace, following two operation modes, namely 1) strict "what you see is what I see" (WYSIWIS), where the visuals are kept identical across all displays, or 2) relaxed WYSIWIS, permitting windows to be of different sizes and by reformatting the text to fit the display nicely [4]. Moreover, in a shared Web page, the scrolling can be independent or synchronized. Sometimes, synchronized scrolling can make the collaboration easier when all users must have the same visual (e.g., during the teacher's expositions). However, independent scrolling allows users to read the same Web page independently in a collaborative work (e.g., during a collaborative information retrieval).

4) Interactive and Temporal Presentations Synchronization: Web pages can have interactive and/or continuous media objects (videos, audios, synchronized multimedia integration language presentations, etc.). In general, synchronization of continuous media presentations at different destinations is not required, but in the case of a cobrowsing system, this appears as an important feature. Synchronized presentations are particularly useful in e-learning environments where, for instance, the instructor can make use of video-based learning objects embedded in Web pages and control the presentation seen by the students in the same fashion as in traditional classes.

5) Ease of Use: A cobrowsing system should be oriented to be used by nonexpert users. Accordingly, it should be simple and easy-to-use, minimizing configuration efforts.

6) Platform Independence: In order to achieve portability, a cobrowsing system should permit its use with general-purpose Web browsers. In this way, a cobrowsing solution allows that users can use their preferred browser in their habitual operational system. GroupWeb [4] Albatross [5], Nestor [6], and Co-Vitesse [7] propose new Web browsers implementing synchronization mechanisms. This approach, however, forces users to use a special Web browser instead of a general-purpose one.

7) Extensibility: Extensible systems present the capability of evolving in the future. In the case of a cobrowsing system, it would be desirable to add new functionalities that enhance the capabilities of the implemented tool.

8) Scalability: To be able to handle a great number of users in cobrowsing sessions without generating excessive network traffic or increasing too much response time. In other terms, the performance of the system should be stable regarding the number of users; a linear progression of the response time with respect to the number of users is, however, acceptable.

B. Currently Implemented Solutions

Three basic approaches for implementing multiuser synchronous cobrowsing systems based on standard Web browsers and servers can be identified.

1) Application Sharing Tools: Such systems enable a user to share his browser with other users during a cobrowsing session. This approach provides strict WYSIWIS. However, changing dynamically the users' synchronization relations requires a great computing and organization effort. Moreover, most

application sharing tools are platform specific, so users are forced to use the same platform. Bandwidth and computer power also play important roles since, normally, in application sharing tools, the actual pixels forming the browser window image are sent over the Internet [2], and on the client computer the image is rebuilt, producing important transmission and performance problems. One example of multiplatform sharing application is virtual network computing [8]. However, using such a solution for providing a cobrowsing service still leads to the bandwidth and performance problems previously identified.

2) Cobrowsing Server-Based Architectures: In this approach, when a user clicks on a link pointing to a specific URL, or when he resizes or scrolls a window in the browser, a browser agent sends the corresponding request to the cobrowsing server. The server then forwards this request to the browsers associated with the users belonging to the same cobrowsing session. An advantage of this approach is that it is not necessary to modify the HTML code for allowing the system to track browsing actions. However, a drawback is that the single cobrowsing server may become a bottleneck leading to performance and reliability limitations for widely distributed workgroups as well as for big-sized groups [9]. Furthermore, a greater drawback of this solution is that it requires the use of signed applets. Many users prefer not to use signed applets due to security risks: they work outside the security sandbox defined for standard applets, so that they have full access to the resources of the system where they are executed.

3) Proxy-Server-Based Architecture: In this approach, a proxy server acts as an intermediate between websites and session members. Any browsing action done by a user may instantly be reflected, via the proxy, to the users synchronized with him. Before sending the requested resources to the users' browsers, the proxy server modifies the HTML code in order to allow tracking future users' browsing actions. Cobrowsing solutions based on this approach are generally easy to deploy because they do not require any plug-in or specific browser installations. The only configuration task that users must perform is to configure their browser to point to the proxy server where the cobrowsing service is provided. This approach is furthermore platform independent. Similar to the cobrowsing server based approach presented above, the use of a single proxy may become a bottleneck and lead to the same performance and reliability problems. To overcome these problems, multiproxy architectures based on distributed cooperating proxies can be adopted. On the other side, the proxy implementation is complex since it may become very difficult to automatically identify, and then modify on the fly, all the code definitions where hyperlinks are specified, especially in Web pages containing JavaScript or JScript code [10].

C. Cobrowsing Tools

In this section, we describe some of the main representative works in the domain of synchronous collaborative Web browsing.

- CWB [10] is an extremely lightweight tool following a cobrowsing server approach, where the cobrowsing server (called Controller Program) is implemented as a Java servlet. It is based on a polling architecture and a dynamic HTML (HTML, CSS, and JavaScript) that avoids intrusive mechanisms based on proxies or events. In terms of cobrowsing organization, a user can run as a master, as a slave, as both, or as neither. One requirement incurred by the exclusive use of JavaScript is that only content that resides on the same Web server as the CWB scripts can be cobrowsed [10]. Therefore, CWB seems to be inadequate for cobrowsing arbitrary websites.
- CoBrowser [11] follows the cobrowsing server approach and works on any platform with a graphical Web browser supporting Java and JavaScript. CoBrowser implements only unmanaged organization, i.e., any user can execute a browsing action, so that the cobrowsing session may become uncontrollable for groups composed of more than three users.
- IMMEX Collaborative [12] includes cobrowsing on a proxy-based approach that also works with any Web browser supporting Java and JavaScript. In terms of session organization, this system allows creating unmanaged and token-passing-based cobrowsing sessions. In a token-passing-based session, only the user that has the token can browse. But any user can decide to take the token (no token retention mechanism is offered). Therefore, IMMEX offers a centralized organization for cobrowsing sections where leadership can be passed to any session member.
- CobWeb [13] is a cobrowsing system where the rules governing the interactions of multiple users (the collaboration protocol) can be externally specified and dynamically modified. It operates with the Netscape browser (using its Java applet capabilities). A CobWeb collaboration server (Java application) also has to be installed and executed on a Unix host to manage group membership. An important drawback of this approach is that, when specifying a protocol, it is necessary to give to the system many parameters, including the set of URLs that are susceptible to be synchronized, which leads to a lack of flexibility for the provided cobrowsing service. Further, the specification of the protocol seems to be a complex task.
- CoWeb [14] follows the proxy approach called CoWeb server. This server translates HTML documents by replacing HTML units through Java applets that enable cooperation. These applets add document-specific cooperative functionality where functions like browsing through the Web, pointing on figures, or filling input fields become cooperative. This system offers different cooperation modes, including the strict WYSIWIS and the relaxed WYSIWIS (called same document). No floor control mechanism of the cobrowsing session is mentioned. Moreover, the proposed translation is not a trivial task.
- PROOF [9] is an extensible proxy-based framework also implemented in Java, where specific application-dependent functionalities have to be implemented in a module to be installed in the proxy. The authors implemented two main application modules for PROOF, enabling two different forms of cooperative browsing, namely: 1) loosely coupled cobrowsing, where each user can browse the Web in an independent way, providing to

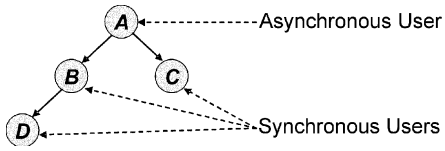

Fig. 1. Basic notion of SDT.

all the users awareness of each others' browsing activity, as well as for users communication; and 2) master-slave browsing, where the superuser of the workgroup assumes the role of master and all the other users (slaves) connected to the workgroup are forced to visit the same Web pages visited by the master. In this cobrowsing system, a page requested by a user is retrieved either from a cache or from the original server specified in the requested URL. In either case, the retrieved resource is parsed by the application module and modified before sending it to the browser as well as to the cache module in the proxy. This proposal was an important source of inspiration when we started developing our own approach.

In this paper, we will propose a new approach for solving the cobrowsing problem, which, when compared to previous systems, presents an important advantage: it provides a simple, very flexible, and powerful synchronization model that allows all the session members to dynamically create and release synchronization relations with other users belonging to the same session.

Furthermore, CoLab deals with the synchronization of embedded audio/video presentations during a cobrowsing session. At the present time, we did not find any cobrowsing application offering such a synchronization scheme or the synchronization of continuous media presentations embedded in Web pages.

III. Synchronization Model of CoLab

In our proposal, we define a CoLab session as a set of users—the session members—engaged in some common browsing activity. In a CoLab session, one or more cobrowsing "workgroups," composed of one or more session members, can exist at the same time. During the lifetime of a session, these workgroups can be dynamically created and destroyed. Two workgroups can be merged into a single one, and a single workgroup can be decomposed into two different workgroups, all that under the initiative of the users.

A. Synchronization Dependency Tree (SDT)

In order to represent the organization of the existing workgroups in a CoLab session, we have chosen to use a data structure called SDT. A typical SDT is shown in Fig. 1.

1) Definition 1: An SDT is a tree structure where nodes represent the users belonging to a single workgroup, and arcs represent the synchronization relations currently existing among them. An arc oriented from node A to node B, where B is the son of A, characterizes the fact that the browsing activities of user B are currently synchronized to those of user A.

2) Definition 2: A single user is called an "asynchronous user" if the node representing him in an SDT is the root node (user A in Fig. 1). This means that this user can choose the Web pages he wants to visualize without any constraint. On the other hand, a single user is called a "synchronous user" if the node that represents him in an SDT belongs to a branch of the tree. In this case, all the browsing activities of this user are synchronized to those of the user at the root of the SDT he belongs to (users B, C, and D in Fig. 1).

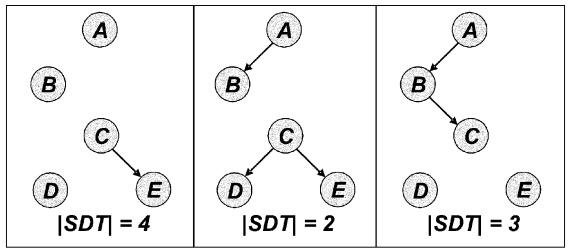

Fig. 2. SDT configuration scenarios.

The tree structure is quite suitable for representing the organization of workgroups in CoLab since: 1) a single user can get synchronized with only one user and 2) several users can be synchronized at the same time with the same user. This is a natural constraint due to the fact that, if we allow creating cross synchronization relations, we will eventually have conflicts between the interests of two or more users having control of the browsing activity.

As we previously said, an SDT is a dynamic structure since the proposed model allows the dynamic creation and release of synchronization relations among connected users. The creation of a synchronization relation leads to binding the Web browsing of a given user to that of another user. Synchronization relations are created and released by using some predefined synchronization primitives. We can understand this approach as an extension of a classical floor control mechanism, where, in the presence of a synchronization relation, the synchronous user looses his floor in favor of the user he gets synchronized with.

At any given moment during a session, depending on the synchronization relations created and released among the connected users, there can exist different numbers of SDTs. This is called the SDT cardinality and represented by |SDT|. This notion is presented in Fig. 2.

As can be clearly seen, we present here three possible synchronization scenarios for users belonging to a session. In the first case, there exists only one synchronization relation, where user E is currently synchronized with user C, while the other users are asynchronous, so $|\text{SDT}| = 4$. In the second case, two new synchronization relations have been created in such a way that now $|\text{SDT}| = 2$. Finally, in the third case, a new synchronization relation has been created, and two others have been released, taking us to a scenario where $|\text{SDT}| = 3$.

The minimal SDT cardinality of a session is 1 when all the users belong to the same SDT, and the maximal cardinality is equal to the number of connected users when all of them are asynchronous, each one representing a single SDT.

B. Synchronization Primitives

CoLab proposes two main synchronization primitives allowing the creation of synchronization relations between users,

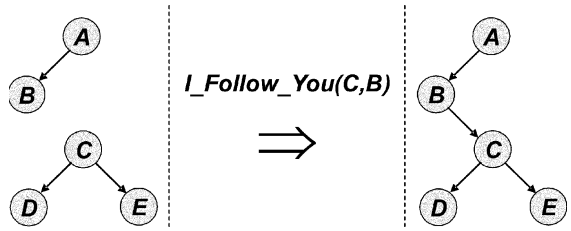

Fig. 3. "I_Follow_You" synchronization primitive.

namely: 1) "I_Follow_You" and 2) "You_Follow_Me."[1] In order to avoid anarchical behaviors, the creation of synchronization relations is subject to an authorization protocol.

The "I_Follow_You" primitive provides the user with the possibility of requesting another user his authorization to get synchronized with him. On the other hand, the "You_Follow_Me" primitive provides a user with the possibility of inviting another user to get synchronized with him. Given that a single SDT node may have several children, the "You_Follow_Me" primitive can be applied to a single user as well as to a set of users.

As previously stated, whenever either of these two primitives is applied, an authorization protocol is started. The user whom the proposal was sent to is asked whether he wants to accept it. If he accepts, the new synchronization relation is created, and the SDTs of the concerned users are merged. Otherwise, no modification is made.

Synchronization relations are released by using the "I_Leave" primitive, which is unconditional: any user involved in a synchronization relation can request it, and it will always succeed. The result of the use of this primitive is that the SDT to which the concerned users belong is split into two single SDTs.

Fig. 3 illustrates the session state before (left) and after (right) the use of the "I_Follow_You" primitive. In the left side of the figure, we can see that $|\text{SDT}| = 2$, where users A and C are asynchronous users, user B is synchronized with user A, and users D and E are synchronized with user C. After the use of the "I_Follow_You" primitive from C to B, both SDTs are merged and become a single SDT whose root is user A, so since that moment the browsing activities of all the users of the session will be synchronized with that of user A.

In Fig. 4, we use an extended state machine-style notation in order to illustrate the general behavior of the synchronization process using the "I_Follow_You" primitive from the point of view of user i. The notation "j!message" represents the sending of the message "message" to user j, and the notation "j?message" represents the receiving of message "message" from user j.

In this figure, the two main states in which user i can be are "async()," when the user is working asynchronously, and "sync()," when the user is synchronized with another user. When user i is in the "async()" state, he can use the "I_Follow_You" primitive on user j. The preconditions to be able to apply this primitive are: 1) user i is asynchronous, and 2) the tree structure is respected. Then, the system passes to an intermediary state where invitation is expressed to the target user and keeps waiting for an answer to the request: an acceptance, a refusal, or an abort. If an abort or a refusal is expressed, user i gets back to the "async()" state, otherwise, the synchronization relation is created, leading, therefore, to user i passing to the "sync()" state. The behavior of the "You_Follow_Me" primitive is symmetric to that of the "I_Follow_You" primitive, so it will not be presented here.

The proposed synchronization model gives CoLab the possibility of supporting the "divide to conquer" concept. The members of a CoLab session are organized into workgroups. Besides, CoLab supports three different organizational structures (based on [15]).

1) Centralized organization, where decisions are made only at the level of the session as a whole (normally implemented by the cobrowsing systems previously presented). It is more adapted to cobrowsing sessions having a leader whose browsing actions must imperatively be followed by the other members (for instance, when a teacher presents a Web-based lecture to a group of students).
2) Decentralized organization composed of different workgroups, where decisions are made independently in each workgroup.
3) Temporarily decentralized organization, which starts out with a decentralized structure and later reintegrates. It is more adapted to cobrowsing sessions where the members can browse independently in order to reach the objectives more quickly (for instance, during a collaborative information retrieval), and whenever they decide, they can get their browsing activities synchronized.

C. Synchronization Model Verification

In order to check the consistency of the use of the proposed synchronization primitives, we have formalized them by using Petri nets. Then, we generated some cobrowsing scenarios and verified that under any circumstance the complete model is consistent.

As a first step, we defined a set of components representing each of the possible behaviors dealing with the creation or release of synchronization relations, as well as the synchronization of the browsing activities executed by the users. Then, we designed a "TCL" script to generate the Petri net and its initial marking, and we used the software tool "TINA" [16] to get the global reachability graph and the tool "CADP" [20] to obtain an abstract view of this reachability graph (a quotient automaton derived from the reachability graph that features only the synchronization primitives; this automaton is observationally equivalent to the reachability graph, see [19] for details). Fig. 5 shows the complete quotient automaton for a session with two users. Other results are available for more users (up to five users, due to the classical state space explosion problem) but are not presented here.

[1]Two other synchronization primitives have been defined to be used by users having assumed privileged roles, namely: 1) "I_Spy_You," which allows a user to get his browser synchronized unconditionally with another user, and 2) "You_Join_Me," which allows a user to "force" other user(s) to get his(their) browser(s) synchronized with him. Their implementation is a future work and is explained in Section VII.

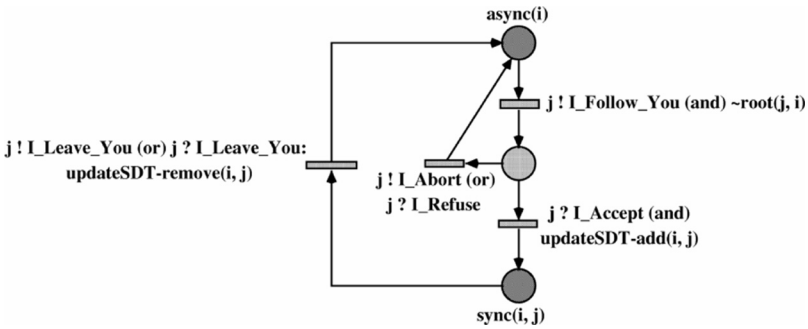

Fig. 4. Synchronization behavior.

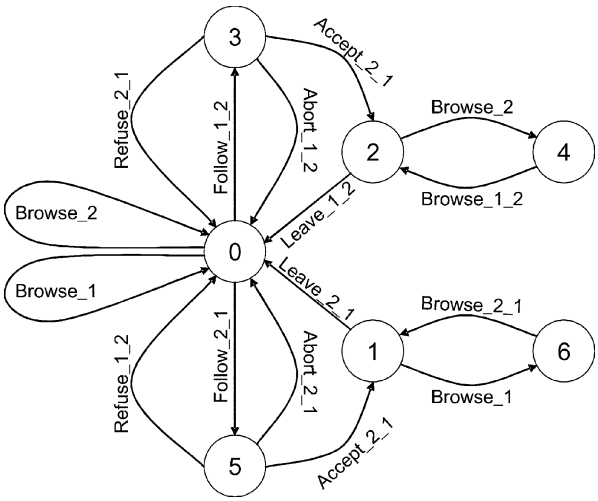

Fig. 5. Complete quotient automaton for a session with two users.

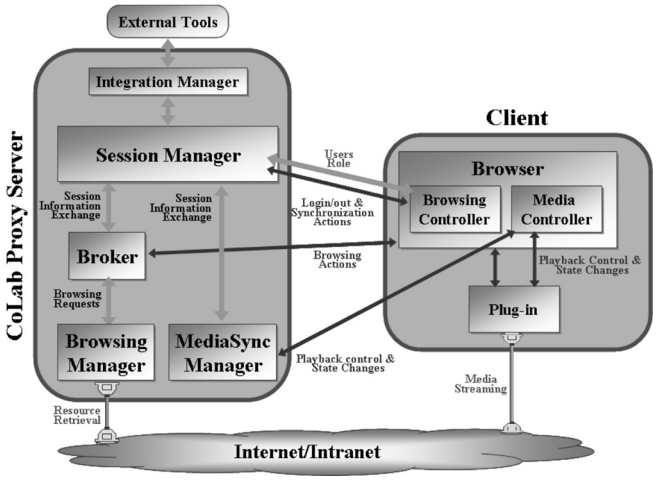

Fig. 6. Collaborative web browsing architecture.

As can be clearly seen, the connected users can be in either asynchronous or synchronous state, and the browsing activity synchronization behavior is consistent with the current synchronization state of the users. The state 0 represents that both users are asynchronous: as a consequence, anyone can browse independently without producing any influence in the browsing activity of any other user (transitions "Browse_1" and "Browse_2"). If, for example, user 1 decides to get synchronized with user 2 (transition "Follow_1_2"), the automaton passes to intermediary state 3 waiting for an authorization, abort, or cancel action. Whenever the creation of the synchronization relation is accepted (transition "Accept_2_1"), the automaton passes to state 2, where whenever user 2 executes a browsing action, user 1 is forced to execute exactly the same browsing action (transition "Browse_2" followed of transition "Browse_1_2"). The part of the "Follow_2_1" is symmetric to the "Follow_1_2," so it will not be explained.

We have analyzed several scenarios similar to the one presented in Fig. 5, and we have been able to formally verify that the synchronization model is fully consistent.

IV. Architecture of CoLab

In Fig. 6, we present the architecture of our cobrowsing system in which we can identify the two main components, namely: 1) the CoLab "proxy server" and 2) the CoLab "client."

A. CoLab Proxy Server

The CoLab proxy server acts as a mediator between the website (where the requested Web pages are hosted) and the users of our system in order to manage cobrowsing sessions. This proxy server is composed of four main modules (Fig. 6), namely: 1) a "session manager"; 2) a "broker"; 3) a "browsing manager"; and 4) a "MediaSync manager." Additionally, it has an "integration manager."

The "session manager" is in charge of managing the cobrowsing session itself. This module offers the authentication and authorization functions based on the cobrowsing session specification defining the default initial page, the available roles[2] and their associated passwords, and the eventual existing privileges that can be associated with each of them.

The main component of the "session manager" is the "synchronization module," which is in charge of treating all the synchronization actions and guaranteeing the overall consistency of the synchronization state. Whenever a synchronization relation is created, the involved users' browsing activities get synchronized as well as the playing of continuous media (eventually embedded in the website).

[2]Access control to CoLab sessions is made in a role-based basis. Roles are to be used in the future as a way to allow some users to have privileges on other users when creating synchronization relations. This is the case of the privileged synchronization primitives "I_Spy_You" and "You_Join_Me."

The "broker" receives any browsing request from the user and asks the "session manager" to verify whether the request should be satisfied. This decision depends on certain conditions, such as the current synchronization state of the user or some other condition specified in an additional module integrated to CoLab (e.g., an access control module).

The "browsing manager" is in charge of all the tasks related to the retrieval of the resources requested by the users. This includes three main components that interact in order to satisfy incoming browsing requests.

1) The "retrieval" module is responsible for retrieving every requested resource. They can be retrieved directly from the Web server specified in the requested URL or from the cache module. In the first case, the retrieval module sends the Web page to the translation module in order to modify it before sending the response to the user's browser, as well as to the cache module.
2) The "translator" module is responsible for modifying on the fly every retrieved Web page. This is necessary to allow our system to track the users' browsing actions. This translation is also required to include the necessary controls for synchronizing the continuous media presentations eventually embedded in these Web pages. The translation consists mainly of adding some control parameters specific to CoLab to each hyperlink definition in the retrieved Web page. When the Web page has embedded media presentations, this module modifies the HTML code in order to detect plug-in state changes and to notify this to CoLab.
3) The "cache" module corresponds to the implementation of a basic cache system, which is mainly used, but not only, for satisfying requests coming from synchronous users in order to improve the performance of the system. We assume that when a synchronous user browses, the requested resource has been previously retrieved by the asynchronous user, so it is faster to retrieve the already translated version of the Web resource directly from the cache rather than retrieving it from the original server and retranslating it at each time.

The "session manager" is also responsible for interacting with the "integration manager" that is intended to provide an API allowing CoLab to be extended with new functionalities, such as an access control system, or to be integrated to other collaborative tools or integration environments, such as LEICA [17].

The "MediaSync manager" takes charge of all the tasks related to the presentation control of the eventual continuous media presentations embedded in Web pages. Its main function is to guarantee the synchronization of audio/video presentations (streamed or downloaded) by forcing the same presentation state in all synchronous users' plug-in. As detailed in [18], this module maintains the current state of each continuous media presentation in the session based on "state change" messages sent by the "media controller" and controls the presentations states by sending "playback control" messages to the synchronous users.

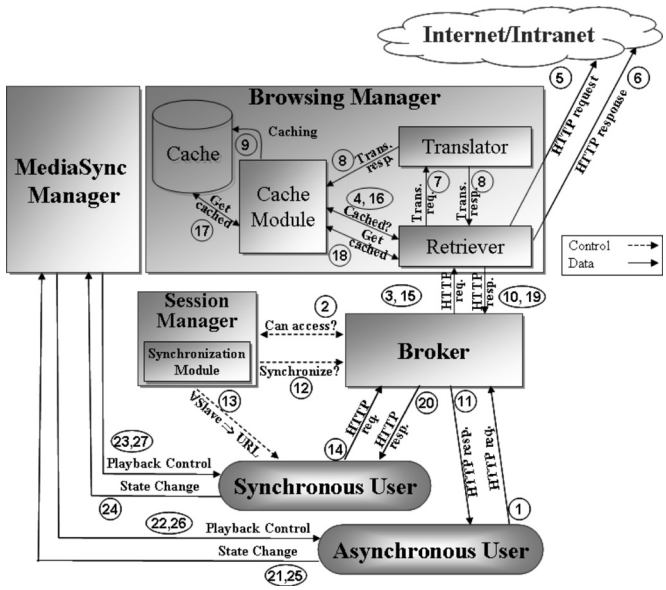

Fig. 7. Synchronization of the browsing and media presentation actions.

B. CoLab Client

The "browsing controller" and the "media controller" are the two main modules present at the client side (see Fig. 6).

The "browsing controller" is the component in charge of establishing a connection with the CoLab proxy server. Through this connection, the users' browsers receive the commands to display Web pages whenever they are synchronized with another user. The "browsing controller" also provides users with all synchronization controls allowing creating and releasing synchronization relations.

The "media controller" controls and synchronizes the continuous media presentation in the current Web page for all users of a workgroup. This module does the following:

- records the state of each audio/video presentation;
- captures state changes of the continuous media presentations, treats them locally, and then informs them to the "MediaSync manager";
- receives playback control messages from the "MediaSync manager" and executes the playback control.

The "media controller" prevents synchronous users from executing any playback control action. In this case, the playback control is done by the "MediaSync manager" via "playback control" messages.

V. Operational Behavior of CoLab

In order to graphically illustrate the operational behavior of our proposal, we present in Fig. 7 the case of a typical browsing action performed by an asynchronous user and the resulting synchronization with another user.

The first step consists of the request of a resource expressed by a user (1), which is treated directly by the "broker." Next, the "broker" contacts the "session manager" to ask it whether the user can retrieve the requested resource (2). If so, the "broker" sends the request to the "retriever" (3), which asks the "cache module" if that resource is already in the cache (4). Let us assume that this is not the case, so the resource is retrieved

directly from the original Web server (5–6), and if it is identified as an HTML resource, 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" for storing purposes (8–9). The "retriever" then sends the resource back to the "broker" (10), which sends it to the user who 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 who are currently synchronized with the user who has just executed the browsing action (12). Then, the "session manager" sends a message to the browser of every synchronous user present in the same workgroup (13). Each browser will then separately make its own request for the indicated resource (14), which will be sent again to the "broker." The "broker" asks the "retriever" (15) for the retrieval of the resource, which itself asks the "cache module" to verify whether the resource is cached (16). Since the resource has already been stored in the cache, and this browsing action is the consequence of the synchronization of a browsing action, it is retrieved directly from the cache (17) and sent back to the "retriever" (18), which sends it back to the "broker" (19), finally satisfying the user's request (20).

Fig. 7 also shows the behavior of the playback control of continuous media presentations. For instance, once a Web page containing a continuous media presentation is loaded, if the asynchronous user clicks the "play" button, the "media controller" sends the state change (21) to the "MediaSync manager." The MediaSync manager module updates the presentation state and sends all users the "prepareToPlay" playback control (22 and 23). When this module receives a "state change" message from all users (24 and 25) indicating the "readyToPlay" state, it sends all users the "play" playback control (26 and 27).

VI. CoLab's Current Implementation

At the present time, we have developed CoLab version 2.0, which implements almost all the concepts presented in this paper. The implementation is fully operational and has been tested under real operational conditions. This section presents some implementation aspects of this prototype.

A. CoLab Proxy Server and CoLab Client

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 1.4.2_09, 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 case of the "media controller" module, at the present time, it only works with the Netscape browser and RealPlayer. RealPlayer was chosen since it implements an access API that provides methods to set and retrieve presentation attributes, control clip playback, and handle user interactions.

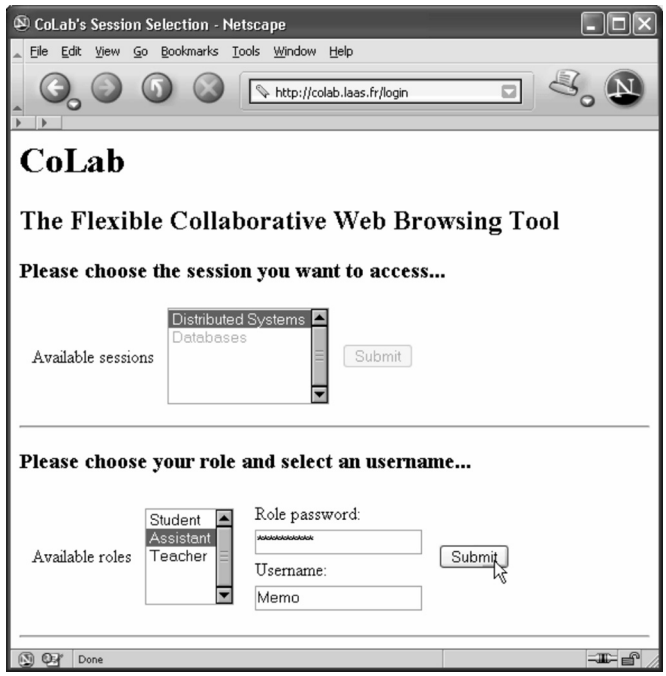

Fig. 8. CoLab's login page.

B. CoLab Current Operational Implementation

In order to use CoLab, the first step to be accomplished is to configure the available cobrowsing sessions. Configuration is done via an extensible markup language (XML) file that contains the specification of the default initial page, the available roles and their associated passwords, and the eventual existing privileges that can be associated with each of them. This file is interpreted by the server when it is started.

An administrator can configure new sessions as well as delete existing ones through a Web application. This is done through a Web service interface defined in the CoLab proxy server. When creating a session, CoLab automatically generates the configuration file and stores it. On the browser side, users must only configure the PAC facility in order to make the browser point to the URL where the configuration file is located. This file contains a JavaScript program that allows dynamically redirecting browsing requests depending on certain criteria. It is actually through this facility that we are able to use any general-purpose Web browser in our platform.

When accessing CoLab, the first step users must perform is to choose a session from those available, choose the role he wants to play, enter the password associated with this role, and select a username, through which he will be identified within the session. The login screen of CoLab is presented in Fig. 8.

Once the user has been authenticated, a new browser window opens: the one where the cobrowsing activity will take place. The other browser window can be used by the user for browsing the Web outside any CoLab session (i.e., no synchronization is possible in this window).

The CoLab session window has two frames: one frame contains the "control frame" presenting browsing and synchronization controls as well as the users' awareness information, and the other frame contains the "Browsing Frame," where

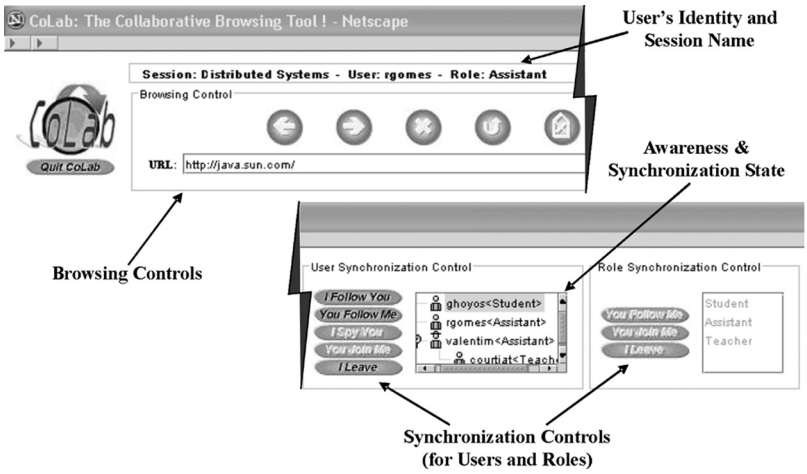

Fig. 9. CoLab's control frame.

the browsed pages will be displayed. A screen capture of the control frame is presented in Fig. 9.

The control frame is composed of a GUI containing all the components that make collaborative Web browsing possible. On the left side (presented at the top of Fig. 9) are the "browsing controls," which are equivalent to those of a typical browser, but here the "Home" button will load the default initial Web page for the session. On the right side (presented at the bottom of Fig. 9) are the "user awareness and synchronization controls." The following are presented: the connected users and their current synchronization state, and the available synchronization controls associated with the synchronization primitives. Here, we can see which users are currently present in the session, and which are the existing synchronization relations among them. For example, in the image presented in the last figure, we can see that there are four users currently logged in the session, namely: 1) "ghoyos"; 2) "rgomes"; 3) "valentim"; and 4) "courtiat"; and we can also see that user "courtiat" is currently synchronized with user "valentim." Concerning the synchronization controls, we can see that they are divided in two sections, namely: 1) "user synchronization" and 2) "role synchronization." The first one contains the buttons representing the synchronization primitives that can be applied to single users, while the second only presents buttons corresponding to the synchronization primitives that can be applied to groups of users playing the same role.

C. Prototype Limitations

The current implementation, which supports most of the proposed functionalities, has been developed and is currently operational. We have currently implemented the "I_Follow_You," "You_Follow_Me," and "I_Leave" synchronization primitives, allowing the basic synchronization operations and sufficient for implementing most cobrowsing sessions.

Regarding the synchronization of continuous media presentations, the concerned plug-ins used for playback are platform dependent and are external to the browser. Accordingly, in order to be able to coordinate their activities, plug-ins must dispose of an API offering all the necessary functions to control continuous media playback and to handle related events. However, as each plug-in might present its own API specification, CoLab should implement a specific solution for each plug-in. The current version of CoLab supports the playback control of embedded continuous media presentations only for Netscape browser and RealPlayer.

In the current version of CoLab, the synchronization of continuous media presentations always happens in the presentation start or in the presentation resume after a playback control (after a pause, stop, seek, fast-forward, or rewind). Therefore, after the start of the synchronized presentation, the synchronization can get lost due to, for instance, network congestion factors. This problem does not have a trivial solution. In [18], we survey some difficulties concerning the synchronization of collaborative continuous media presentations.

D. Experimental Evaluations

We carried out two experimental evaluations of the CoLab prototype, namely: 1) the performance evaluation and 2) an informal test of the prototype with real users.

The goal of the first experiment was to evaluate the CoLab performance for implementing the cobrowsing activity. For that purpose, we have chosen a set of websites distributed all over the world and have then executed an application that simulates a user who gets connected to a specific session and then gets synchronized with a predefined user (in this setting, the asynchronous user is a person that makes some browsing actions). We made several measurements, considering synchronous users groups with sizes from 1 to 165 synchronous users. All the tests have been performed using nondedicated workstations connected through a 100-Mb/s switched LAN. Future developments will specifically address the quality-of-service (QoS) issue within the context of WAN/LAN networks. The following average results have been obtained.

As we can easily see in Fig. 10, the performance of CoLab is quite satisfactory, since, in the worst case with 165 synchronous users in a group, the average retrieval time of a Web page is about 1.2 s.

For using CoLab, there is no *a priori* assumption about the underlying communication network, with the exception that it should offer limited delay to ensure an adequate level of QoS.

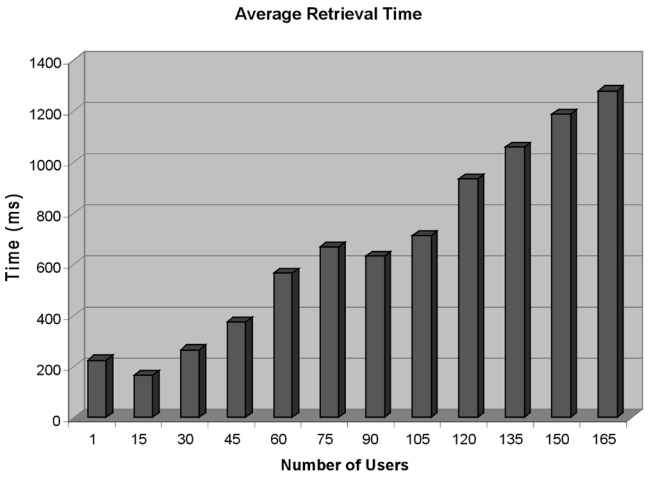

Fig. 10. Performance test of CoLab.

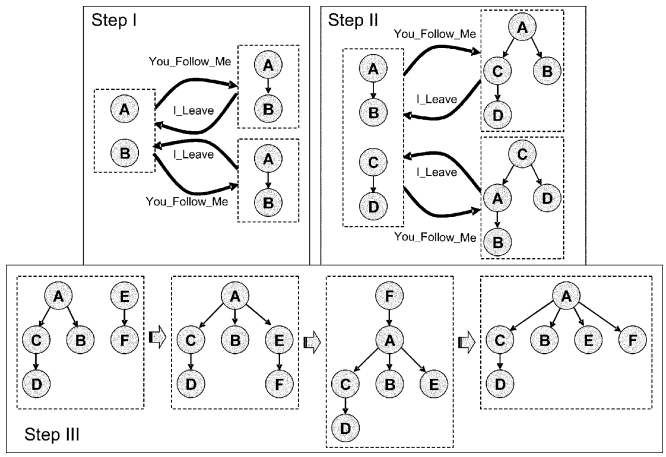

Fig. 11. SDTs generated during the prototype testing.

This is an important issue since CoLab is a synchronous collaboration tool, and large delays may hinder users' coordination.

In terms of bandwidth consumption, we claim that there is practically no overload associated with the use of synchronization primitives. The only information exchanged between the connected clients and the CoLab proxy server consists of short messages associated with the synchronization protocol and the strings containing the URLs to be retrieved to achieve the synchronization of the browsing activities. Furthermore, for synchronous users, resources are directly retrieved from the cache system, reducing network overload.

The second experiment aimed at driving informal testing of the prototype with real users. The main goal of this test is to know if the synchronization mechanism of CoLab ensures that a real cobrowsing session can be accomplished as natural as possible (with minimal effort and confusion). Moreover, we wanted to find out possible malfunctions and ensure trouble-free operation.

The subjects of this test were six computer science undergraduate students at Federal University of Espirito Santo (UFES), Vitoria, ES, Brazil. The task to be accomplished by the subjects was to choose, via cobrowsing, the website containing the best tutorial on computer network topologies from a list of predefined websites. While the CoLab proxy server was installed at Laboratoire d'Analyse et d'Architecture des Systemes (LAAS), Toulouse Cedex, France, all the subjects were in the same room at UFES, each one running a CoLab client in distinct computers.

We controlled the procedure executed by the subjects while accomplishing the imposed task. Our goal was to verify that CoLab effectively supports the implementation of the concept of "divide to conquer." During the execution of the task, all synchronization operations realized by each workgroup were registered and some of them are presented in Fig. 11.

The procedure adopted for this informal test started by a brief introduction to the CoLab tool and to the test itself. Then, the following steps were accomplished:

Step I: The group was divided into three workgroups composed of two members. Each workgroup should cobrowse the list of websites in order to analyze the presentation quality regarding one specific network topology (i.e., initially, each group was in charge of a different network topology). The first frame in Fig. 11 schematizes the use of different synchronization primitives by the members of one workgroup. Actually, we had the same synchronization patterns for the three workgroups: users have interchangeably used the different synchronization primitives, several times, so as to alternate the synchronization relationship between them. This change of synchronization control is possible thanks to the available synchronization primitives addressed to other members. Note that this change is possible only if the actual asynchronous member accept this change. This frequent changing of synchronization control allowed the two members to cobrowse in a cooperative fashion.

Step II: After finishing Step I, the first two groups were merged into a new group, responsible now for choosing its preferred sites regarding the two topologies initially analyzed by each one of the groups. The third workgroup was temporally disconnected from the session. As presented in the second frame of Fig. 11, in this step, users decided to organize themselves as if they were two workgroups. Then, the asynchronous members of each workgroup (A and C) have interchangeably used the synchronization primitives. This way, they could alternate the guiding of the browsing activity while they presented the analysis of the respective workgroup about the browsed Web sites (B and D stayed as observers).

Step III: Finally, all members joined together in order to choose the best tutorial covering the different network topologies. On the bottom of Fig. 11, we present some snapshots of the SDT evolution during the cobrowsing activity. As illustrated by this SDT evolution, the member A was in charge of presenting the analysis resulting from the previous step to the members of the third workgroup (which did not participate in Step II). A controls the conavigation at the beginning of this step, but then one of

the members of the third workgroup (F) take the control during a period of time in order to expose the analysis of this workgroup. In the end of this step, member A takes control to conclude the task.

Regarding the synchronization operations realized by the workgroup members, we can conclude that the prototype effectively allowed the dynamic creation and merging of workgroups and, therefore, provided means for "divide to conquer."

After completing the task, the subjects filled out a questionnaire to rate various aspects related with the effectiveness of the prototype. Moreover, members of the group were interviewed about the experiment.

From the questionnaires and interview, we could confirm that the prototype allowed to adequately accomplish the task. The group considered that the interface of CoLab is simple and easy to use. Some minor problems were pointed out (which we will take in consideration in the next version of CoLab).

The synchronization primitives were considered relatively easy to use and intuitive. The group considered that CoLab implements a great part of functions necessary for cobrowsing. Some students pointed out that the use of two primitives ("I_Leave" followed by "You_Follow_Me") is not very natural when a synchronous member of a workgroup desires to become the asynchronous member of this workgroup (i.e., the user wants to take the browsing control of the workgroup he belongs to). As a result, we will include in the CoLab interface a button allowing such a synchronization request (representing a composition of the actual synchronization operators of our synchronization model).

VII. Conclusion and Future Work

In this paper, we have defined a general-purpose proxy-based collaborative Web browsing system called CoLab. We claim that this system gives the users a great flexibility for establishing collaboration relations while browsing, creating in this way an environment where collaboration is greatly facilitated. Our model meets most of the basic requirements for a system aimed at supporting generic synchronous cobrowsing applications.

1) Session Management: CoLab permits the dynamic creation of cooperative multigroup multiuser sessions and manages the list of users in each session. Authenticated users can join or leave a session at any time. Several user roles can be associated with a session.
2) Synchronization Management: CoLab offers the possibility of easily creating and releasing browsing synchronization relations. Moreover, it provides a new paradigm since a cobrowsing session can adopt either a centralized, a decentralized, or a temporally decentralized organization. The capacity of creating cobrowsing sessions composed of workgroups that can at any time be reorganized dynamically turns CoLab into a very flexible and useful tool.
3) Web Browsing Synchronization: CoLab implements only relaxed WYSIWIS. Moreover, in a shared Web page, the scrolling is independent, allowing users to look at the same Web page independently in a collaborative work.
4) Interactive and Temporal Presentations Synchronization: CoLab allows synchronization of continuous media presentations. The current prototype supports the playback control of embedded continuous media presentations only for the Netscape browser and RealPlayer.
5) Ease of Use: CoLab is a platform oriented to nonexpert users because it requires neither plug-in nor special browser installations. Users must only configure their browser load the CoLab's PAC file.
6) Platform Independence: CoLab can be used with any standard Web browser. The only requirement is that it supports Java and the PAC facility.
7) Extensibility: Extensions can easily be brought to CoLab thanks to its integration API.
8) Scalability: One potential problem of the current CoLab architecture is related to its scalability, in particular at the level of the proxy server. In the current implementation, we consider a centralized implementation and have shown that, at least up to 165 synchronous users, the average retrieval time of a Web page is linear with respect to the number of users in a workgroup. Further studies dealing with the distribution of the proxy 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.

CoLab follows that the proxy server architecture and the proxy program are complex: it is not a trivial task to detect all of the ways in which the HTML and scripts in an arbitrarily shared Web page might create hyperlinks. We will then study the possibility of treating every kind of resource in a format other than HTML (i.e., JavaScript, JScript, Flash, etc.).

In the future, we will keep working on the implementation of all the features of the proposed model (i.e., the "I_Spy_You" and "You_Join_Me" synchronization primitives, roles, and privileges). Roles are intended to be used to group users with similar characteristics, allowing us to assign special privileges to certain users having assumed a given role on other users having assumed another role. In this way, we can allow users having assumed a privileged role to apply any of the two new unconditional synchronization operators ("I_Spy_You" and "You_Join_Me") on other users. We will work as well in 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 have already started the implementation of the "integration" API, identifying possible requirements for integrating CoLab with other collaborative systems and tools, as well as to add new functionalities to CoLab. In the particular case of the integration of CoLab to other collaborative tools, we are currently working on the integration of CoLab to a multiroom Chat based on LEICA [21], an environment for integrating collaborative applications in a transparent way.

Another subject on which we will start working soon is the implementation of the distributed version of our platform (distributed proxies instead of a single proxy) in order to avoid any bottlenecks in the presence of heavy workload. Moreover,

other continuous media presentation plug-ins (in addition to RealPlayer) will be investigated, and those providing APIs allowing to set and retrieve presentation attributes will be incorporated in the CoLab system. Finally, complementary synchronization semantics will be studied.

ACKNOWLEDGMENT

The authors would like to thank all the computer science undergraduate students from Federal University of Espirito Santo (UFES), Vitoria, ES, Brazil, who participated in testing the CoLab prototype.

REFERENCES

[1] N. C. Romano, Jr., D. Roussinov, J. F. Nunamaker, and H. Chen, "Collaborative information retrieval environment: Integration of information retrieval with group support systems," in *Proc. 32nd Hawaii Int. Conf. Syst. Sci.*, 1999, vol. 1, p. 1053.

[2] J. K. Lin. (2005). *An Insider's Guide to Today's Cobrowsing Technologies, white paper of PageShare Technologies Inc.* [Online]. Available: http://www.pageshare.com

[3] G. J. Hoyos-Rivera, R. L. Gomes, and J. P. Courtiat, "A flexible architecture for collaborative browsing," in *Proc. 11th IEEE WetICE, Workshop Web-Based Infrastructures and Coordination Architectures Collaborative Enterprises*, Pittsburgh, PA, 2002, pp. 164–169.

[4] S. Greenberg and M. Roseman, "GroupWeb: A WWW browser as real time groupware," in *Proc. CHI Companion*, Vancouver, BC, Canada, 1996, pp. 271–272.

[5] P.-J. Yeh, B.-H. Chen, M.-C. Lai, and S.-M. Yuan, "Synchronous navigation control for distance learning on the Web," in *Proc. 5th Int. World Wide Web Conf.*, May 1996, pp. 1207–1218.

[6] R. Zeiliger, "Supporting constructive navigation of Web space," in *Proc. Workshop in Personalized and Social Navigat. Inf. Space*, K. Hook, D. Benyon, and A. Munro, Eds., Stockholm, Sweden, Mar. 16–17, 1998, SICS Tech. Rep. T98:02, pp. 91–101.

[7] Y. Laurillau, "Synchronous collaborative navigation on the WWW," in *Proc. ACM CHI*, pp. 209–308.

[8] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper, "Virtual network computing," *IEEE Internet Comput.*, vol. 2, no. 1, pp. 33–38, Jan./Feb. 1998.

[9] G. Cabri, L. Leonardi, and F. Zambonelli, "A proxy-based framework to support synchronous cooperation on the Web," *Softw., Pract. Exp.*, vol. 29, no. 14, pp. 1241–1263, Dec. 1999.

[10] A. W. Esenther, "Instant co-browsing: Lightweight real-time collaborative Web browsing," in *Proc. 11th Int. WWW Conf.*, Honolulu, HI, 2002, pp. 107–114.

[11] K. Maly, M. Zubair, and L. Li, "CoBrowser: Surfing the Web using a standard browser," in *Proc. World Conf. Educational Multimedia, Hypermedia Telecommun.*, 2001, vol. 2001, pp. 1220–1225.

[12] L. Gerosa, A. Giordani, M. Ronchetti, A. Soller, and R. Stevens, "Symmetric synchronous collaborative navigation," in *Proc. IADIS Int. Conf. WWW/Internet*, 2004, pp. 748–754.

[13] D. Stoots, J. Prins, and L. Nyland, "CobWeb: Visual design of collaboration protocols for dynamic group Web browsing," in *Proc. Visual Comput. (Distributed Multimedia)*, San Francisco, CA, Sep. 26–28, 2002, pp. 595–598.

[14] S. Jacobs, M. Gebhardt, S. Kethers, and W. Rzasa, "Filling HTML forms simultaneously: CoWeb—Architecture and functionality," in *Proc. 5th Int. World Wide Web Conf.*, May 1996, pp. 1385–1395. [Online]. Available: http://www5conf.inria.fr/fich_html/papers/P43/Overview.html

[15] N. Siggelkow and D. Levinthal, "Temporarily divide to conquer: Centralized, decentralized," *Org. Sci.*, vol. 14, no. 6, pp. 650–669, Nov. 2003.

[16] B. Berthomieu, P.-O. Ribet, and F. Vernadat, "The tool TINA-construction of abstract state spaces for petri nets and time petri nets," *Int. J. Production Res.*, vol. 42, no. 14, pp. 2741–2756, 2004.

[17] R. L. Gomes, G. J. Hoyos-Rivera, and J. P. Courtiat, "Loosely-coupled integration of CSCW systems," in *Proc. 5th IFIP Int. Conf. DAIS*, Athens, Greece, Jun. 2005, vol. 3543, pp. 38–49.

[18] G. J. Hoyos-Rivera, R. L. Gomes, J. P. Courtiat, and R. Willrich, "Collaborative Web browsing tool supporting audio/video interactive presentation," in *Proc. IEEE 14th Int. Workshops WetICE*, Linkoping, Sweden, 2005, pp. 78–83.

[19] R. Milner, *A Calculus of Communicating Systems*, vol. 92. New York: Springer-Verlag, 1980.

[20] Construction and Analysis of Distributed Processes. (2006). [Online]. Available: http://www.inrialpes.fr/vasy/cadp/

[21] G. J. Hoyos-Rivera, R. L. Gomes, and J. P. Courtiat, "CoLab: Co-navigation sur le Web," in *Proc. NOuvelles TEchnologies de la REpartition (NOTERE)*, Toulouse, France, Jun. 2006. in press.

Guillermo de Jesús Hoyos-Rivera received the B.S. degree in informatics and the M.Sc. degree in artificial intelligence from the Universidad Veracruzana, Xalapa, México, in 1992 and 1997, respectively, and the Ph.D. degree from the Université Paul Sabatier, Toulouse, France, in 2005. In 1999–2000, he headed the Artificial Intelligence M.Sc. program with the Universidad Veracruzana.

Since 1997, he has been a full-time Researcher with the Universidad Veracruzana. His research interests are Web technologies, parallel and distributed computing, computer networks, and software agents.

Roberta Lima Gomes received the B.Sc. degree in computer engineering from the Federal University of Espírito Santo (UFES), Vitoria, ES, Brazil, in 1999, the M.Sc. degree from the Federal University of Rio de Janeiro, Rio de Janeiro, Brazil, in 2001, and the Ph.D. degree from Université Paul Sabatier, Toulouse, France, in 2006.

She is currently a Professor with the Computer Science Department, UFES. Her research interests lie in computer-supported cooperative work systems, distributed systems, and multimedia communication.

Roberto Willrich received the B.S. and M.Sc. degrees in electrical engineering from the Federal University of Santa Catarina, Florianopolis, SC, Brazil, in 1988 and 1991, respectively, and the Ph.D. degree from Université Paul Sabatier, Toulouse, France, in 1996.

He is currently a Professor of computer science with the Federal University of Santa Catarina. He is currently in a postdoctoral position at the Laboratoire d'Analyse et d'Architecture des Systemes-Centre National de la Recherche Scientifique (LAAS-CNRS) supported by a CAPES-COFECUB (Brazil) grant. His research interests are in the areas of multimedia computing and communication, quality-of-service, and Web technologies.

Jean-Pierre Courtiat received the degree in computer science from the École Nationale Supérieure d'Électronique, d'Électrotechnique, d'Informatique, d'Hydraulique, et de Télécommunications, Toulouse, France, in 1973, and the Ph.D. degree and the "Doctorat d'Etat" degree in computer science from the University of Toulouse, Toulouse, in 1976 and 1986, respectively.

From 1973 to 1976, he was a Researcher with the Laboratoire d'Analyse et d'Architecture des Systemes, Centre National de la Recherche Scientifique (LAAS-CNRS). From 1976 to 1980, he was a Professor of computer science with the Federal University of Rio de Janeiro, Rio de Janeiro, Brazil. In 1980, he returned to LAAS as a "Chargé de recherche au CNRS" and then a "Directeur de recherche au CNRS." His research interests include design of protocols; definition and application of formal methods to the specification, verification, and testing of protocols; distributed systems; and multimedia applications. He contributes to several researches and has also participated in standardization activities, as an expert of AFNOR and ISO.

Dr. Courtiat is a member of the Association for Computing Machinery.