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 in order 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.

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. Synchronization relies on a simple authorization protocol through which users can invite other users to create and release browsing synchronization relations. Additionally, special synchronization operators, intended to be used by privileged users, are also proposed with the purpose of overriding the authorization protocol.

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

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). At the present time users are isolated when browsing the Web 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. We understand a synchronization relation as binding the Web browsing of one user, called a synchronous user, to that of another one. In this way the browser of the synchronous user will automatically retrieve and present the same Web pages sequence than that requested by the user he is synchronized with.

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. This adds a new dimension 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 the related work, 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 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.

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 for 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 for collaboratively indexing bibliographical databases. While 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 resources currently being browsed by users, which ensures awareness among users browsing similar documents, facilitating therefore their interaction.

- **E-CoBrowse** [4] is a system that uses Chatpointer (chat-style conference tele-pointers) allowing users to communicate while browsing.

- **Let’s Browse** [5] is an agent to assist a group of people in browsing, by suggesting new material likely to be of common interest. It is an extension to the Web browsing agent Letizia, which does a real-time, incremental breadth-first search around the user's current page, filtering 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 requested resources depending on the type of device used for accessing the Web (PC, Laptop, PDA, etc.).

- **WikiWikiWeb** [8] is a system that contains a collection of Web pages. It allows 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, communicate and interact by the use of several available communication tools. All of them include a synchronized browsing facility.

All these proposals offer alternatives for collaboratively browsing the Web in some degree. The main difference of our approach with respect to these ones is that we focus on the dynamic creation and release of synchronization relations among the users. The creation of synchronization relations, 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 synchronous user looses his floor in favor of the user he is synchronized with.

For creating synchronization relations an authorization protocol is used. This 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 keeps unchanged. Additionally we consider also including 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 present the architecture of our Collaborative Web Browsing system, which is graphically presented in Figure 1.

![Collaborative Web Browsing Architecture](image)

**Figure 1. Collaborative Web Browsing architecture**

Here 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 of tracking the connected users’ activities, 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 the requested resources. This is not directly accessed by the users, but though the **Broker**. The reason is that browsing requests are not to be systematically satisfied, but it depends 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 it should be satisfied.

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

The Synchronous User component is intended to allow our environment to track the browsing actions. The Session Manager sends a message to every single Browser to synchronize this browsing action executed by a user, and the resulting synchronization of this to other user.

Figure 3. Synchronization of the browsing actions

3.1. Operation behavior of CoLab

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

The first step consists in 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), who asks the Cache Module if that resource is already in the local cache (4). Let’s 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 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 (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). Here the Session Manager sends a message 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).
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 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. The other users belonging to the same SDT are called synchronous users. The basic notion of SDT is presented in Figure 4. Here the direction of the arrow denotes direction of the propagation of the synchronization. In order to illustrate these notions we present in the Figure 5 some possible configuration scenarios of the SDTs. The number of SDTs actually existing in a session is the number of connected users, when all of them are working asynchronously. The minimum number of SDTs in a session is one, when all the users belong to the same SDT. The number of SDTs actually existing in a session is called the SDT Cardinality, denoted by |SDT|. In order to illustrate these notions we present in the Figure 5 some possible configuration scenarios of the SDTs in a session.

![Figure 4. Basic notion of SDT](image)

At any moment in a session there may be as many SDTs as asynchronous users. Whenever two users decide to get synchronized their SDTs merge becoming a single SDT. The maximum number of possibly existing SDTs in a session is the number of connected users, when all of them are working asynchronously. The minimum number of SDTs in a session is one, when all the users belong to the same SDT. The number of SDTs actually existing in a session is called the SDT Cardinality, denoted by |SDT|. In order to illustrate these notions we present in the Figure 5 some possible configuration scenarios of the SDTs in a session.

![Figure 5. SDTs configuration scenarios](image)

The first scenario represents the case when every user is working asynchronously, so |SDT|=5. As users create or destroy synchronization relations, the number of existing SDTs changes. In the second scenario we can see that users C, D and E have got synchronized with user B. In this case |SDT|=2. In the last scenario we can see that a single SDT may in fact have several levels. In this case user B has decided to get synchronized with user A, so |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. Given that a single SDT node may have several children, the You_Follow_Me operator 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 or not to accept it. Synchronization relations are released by using the operator I_Leave, which is unconditional: 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 6 we use extended state machine-style notation.

![Figure 6. Synchronization behavior](image)
unconditionally synchronized with another user. In this case the only user in the 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_13, 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 7 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 7: CoLab’s implementation architecture](image)

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/ for the access to Web pages, and RTP/UDP/ for the continuous streams). Current experimentations are carried out on a 100 Mbps switched Etherent LAN, however future developments will address specifically the QoS issue within the context of WAN/LAN networks.

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

The first step in order to use CoLab is to configure the available sessions. Configuration is done via an XML file which contains the specification of the default initial page, the available roles and their associated passwords, and the eventual existing privileges that can be associated to each of them. Once the sessions have been configured, the CoLab server can be started.

On the browser side, users must configure the Proxy Automatic Configuration facility in order to point the place where the configuration file is. This file contains a JavaScript™ program that allows dynamically redirecting browsing request depending on certain criteria.

When accessing CoLab, the first step is to choose a session from those available, then the user must choose the role he wants to assume, type the password associated to this role, and choose a username. The login screen is presented in the Figure 8.

![Figure 8: CoLab’s login page](image)

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 9.

![Figure 9. The Control Frame](image)

The Control Frame contains a GUI containing all the components that 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, but here 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 (presented at the bottom of Figure 12) 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. 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 the possibility of easily creating and releasing browsing synchronization relations. We think that this orientation gives the users a lot of flexibility for establishing collaboration relations while browsing, creating in this way an environment where collaboration is greatly facilitated.

The current operational implementation, supporting only a subset of the whole functionality, has been developed and it is currently operational. 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


