A Service-Oriented Architecture for Collaboratively Browsing the Web

Guillermo de Jesús Hoyos-Rivera[1], Giner Alor-Hernández[2], Roberta Lima Gomes[3], Roberto Willrich[4] and Jean-Pierre Courtiat[5]

[1]Department of Artificial Intelligence, School of Physics and Artificial Intelligence Universidad Veracruzana, Xalapa, Veracruz

[2]Division of Research and Postgraduate Studies, Instituto Tecnologico de Orizaba, Orizaba, Veracruz

[3]Informatics Department, Universidade Federal do Espírito Santo, Vitoria, Brasil

[4]Informatics and Statisticis Department, Universidade Federal de Santa Catarina, Florianópolis, Brasil

[5]Laboratory of Analysis and Architecture of Systems-CNRS, Toulouse, France

ghoyosr@gmail.com, galor@itorizaba.edu.mx, rgomes@inf.ufes.br, willrich@inf.ufsc.br, courtiat@laas.fr

Abstract. Service-Oriented Architectures (SOA) transform the ways in which the applications are created in a distributed environment work. Commonly, these applications are based on Web Services technologies. Web Services allow integration and collaboration through Internet standards. Recently, SOA has emerged as paradigm to develop collaborative systems, specially, Collaborative Web Browsing. Collaborative Web Browsing (co-browsing) aims at extending currently available Web browsing capabilities in order to allow several users to "browse together" on the Web. A co-browsing 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 co-browsing system called CoLab from a point of view of SOA. CoLab provides all the functionalities required for allowing users to collaboratively browse the Web. CoLab presents a service-oriented architecture where the functionalities for co-browsing are performed.

Keywords: Collaborative Web browsing, continuous media, Service-Oriented Architecture, synchronization, Web Services.

1 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.

© *S. Torres, I. López, H. Calvo. (Eds.)*
Advances in Computer Science and Engineering
Research in Computing Science 27, 2007, pp. 305-317

Received 23/02/07
Accepted 08/04/07
Final version 21/04/07

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 activities (i.e. the pages they are visiting). Collaborative Web browsing overcomes this problem by allowing users to "browse together". In this paper, we consider a co-browsing system as a tool for allowing users to browse Web pages together in co-browsing sessions while establishing/releasing browsing synchronization relations among them as they wish.

This co-browsing system is based on a Service-Oriented Architecture (SOA) which describes a software architecture that defines the use of loosely coupled software services. Within our SOA, we provide a brokering service which uses Web Services technologies and can interoperate with other systems or software agents. We believe 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.

However, there are several requirements that a co-browsing solution must meet. We believe that one of the most important ones is to provide flexible capabilities for organizing co-browsing 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 co-browsing solutions adopt two types of organization for a co-browsing 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 co-browsing 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 co-browsing sessions where the browsing actions of the leader must be followed by all the other session members. An alternative proposal for the organization of co-browsing sessions is allowing dynamic organization of session.

Here, session members can dynamically reorganize the co-browsing 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" [1], which is very important in Computer Science.
In this paper, we propose a co-browsing system called CoLab [2]. This co-browsing system is based on a simple and powerful synchronization model supporting a dynamic organization of a co-browsing session. The proposed model offers a simple mechanism allowing session members to create and release synchronization relations among them.

2 Service-Oriented Architecture of CoLab

SOA is a new approach to application development that requires people to work and think more cohesively and collaboratively than before. SOAs are based on the notion of services, which are high-level software components that include Web Services.

Web Services have attracted a lot of attention over the past years as a means of building and deploying software to simplify development and systems integration. Web Services are ideal for application integration and collaboration of internal systems or for linking software components over the Internet. Web Services technologies are based on open standards recommended by the World Wide Web Consortium (W3C). In this sense, we propose a co-browsing system following the SOA basic principles: 1) Integration, 2) Discovery and, 3) Publish. This co-browsing system is called CoLab. Some internals of CoLab are built on SOA. CoLab delivers a collaborative environment that includes capabilities such as co-browsing session, document and Web content management. In the service-oriented architecture of CoLab, there are two main components: 1) the CoLab "proxy server" and 2) the CoLab "client" which are described below. The general architecture is depicted in Fig. 1. 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 co-browsing sessions.

This proxy server is composed of four main modules 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" manages the co-browsing session itself. This module offers the authentication and authorization functions based on the co-browsing session specification defining the default initial page, the available roles and their associated passwords, and the eventual existing privileges that can be associated with each of them. Roles are used as a way to allow some users to have privileges on other users when creating synchronization relation. The main component of the "session manager" is the "synchronization module," which treat 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).

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 broker is proposed to be implemented as a Web Service-based brokering service. The general architecture of this module is depicted in Fig. 2. The broker is built by the following components:

1. **Service Registry** is the mechanism for registering and publishing information about services supported by CoLab. In this sense, we used a private UDDI [3] node which is an industry initiative to create a platform-independent, open framework for describing, discovering, and integrating Web Services.

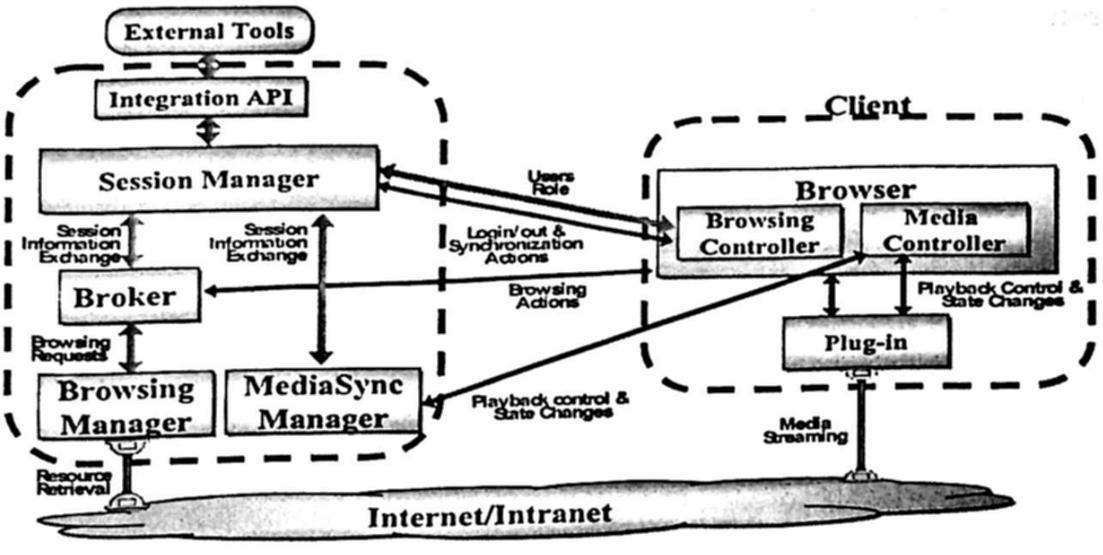

Fig. 1 Collaborative web browsing architecture.

2. **Discovery Service** is a component used to discover Web Services implementations. These Web Services can be obtained from the private UDDI node. Inside the discovery service, there is a query formulator which builds queries that will be sent to the registry service. This module retrieves a set of suitable services selected from the previous step and creates feasible/compatible sets of services ready for binding. The discovery service uses sophisticated techniques to dynamically discover Web Services and to formulate queries to UDDI nodes.
3. **Dynamic Binding Service** is a component that binds compatible Web Services. The binding of a Web Service refers to how deep is the degree of coupling with other Web Services. For instance, the technology of one Web Service provider might be incompatible with that of another even though the capabilities of both of them match with some requirements. In this sense, the module acts as an API wrapper that maps the interface source to a common interface supported by CoLab.
4. **Dynamic Invoker** transforms data from one format to another. This component can be seen as a data transfer object which contains the data (i.e. request or response) flowing between the requester to the provider applications of Web Services. We propose the use of Web Services Invocation Framework (WSIF) that is a simple Java API for invoking Web Services, no matter how or where the services are provided [4]. WSIF allows stubless or completely dynamic invocation of a Web service, based upon examination of the meta-data about the service at runtime. It also allows updated implementations of a binding to be plugged into WSIF at runtime, and it allows the calling service to defer choosing a binding until runtime.

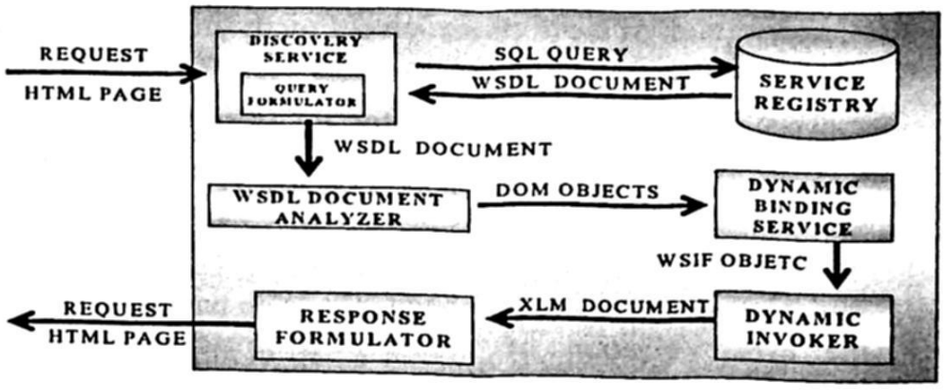

Fig. 2 Brokering service architecture.

5. **WSDL Document Analyzer** validates WSDL documents. In this context, this component reports the operations, input and output parameters, and their data types in a XML DOM tree. We propose the use of WSDL4J [5] to convert the XML DOM nodes in Java objects. It facilitates the creation, representation and manipulation of WSDL documents. WSDL4J API is an IBM reference implementation of the JSR-110 specification (Java API's for WSDL).
6. **Response Formulator** receives the responses from the suppliers about a requested product. This module retrieves useful information from the responses and builds a XML document with information coming from the service registry and the invocations' responses. This XML document is presented in HTML format using the Extensible Stylesheet Language (XSL).

The "browsing manager" carries out 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 [6]. 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 [7], 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.

The "browsing controller" and "media controller" are two modules present at the client side (see Fig 1). 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 three functions:

1) records the state of each audio/video presentation;
2) captures state changes of the continuous media presentations, treats them locally, and then informs them to the "MediaSync manager";
3) 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.

In next section, we present the operational behavior of CoLab through a typical case of study.

3 Operational Behavior of CoLab

In order to graphically illustrate the operational behavior of our proposal, we present in Fig. 3 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. 3 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." TheMediaSync 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 "ready-ToPlay" state, it sends all users the "play" playback control (26 and 27).

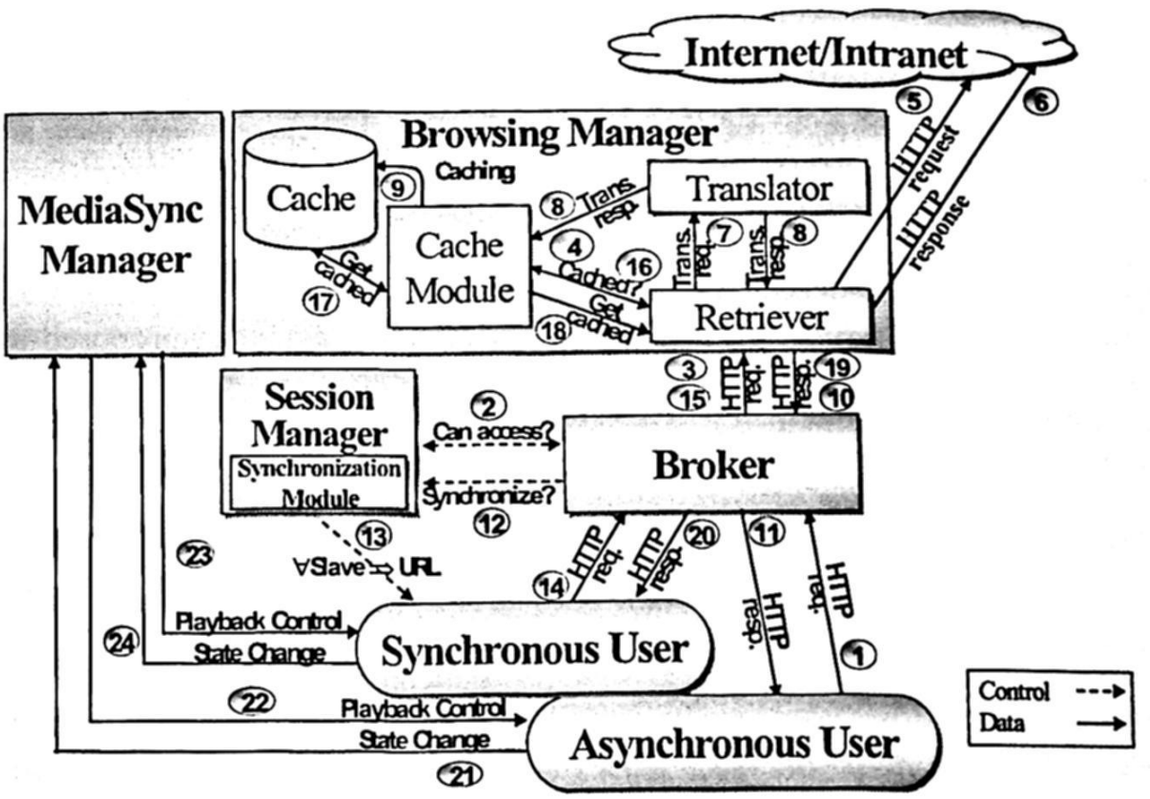

Fig. 3 Synchronization of the browsing and media presentation actions.

4 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 co-browsing "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 split into two different workgroups, all that under the initiative of the users.

Synchronization Dependency Tree (SDT)

In order to represent the organization of workgroups in a CoLab session, we have chosen to use a data structure called SDT. A typical SDT is shown in Fig. 4a.

1) **Definition 1:** A 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. 4a). This means that this user can freely decide his browsing activities. 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 or leaf of the SDT. 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. 4a).

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. 4b. 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 $|SDT| = 4$. In the second case, two

new synchronization relations have been created in such a way that now |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 |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.

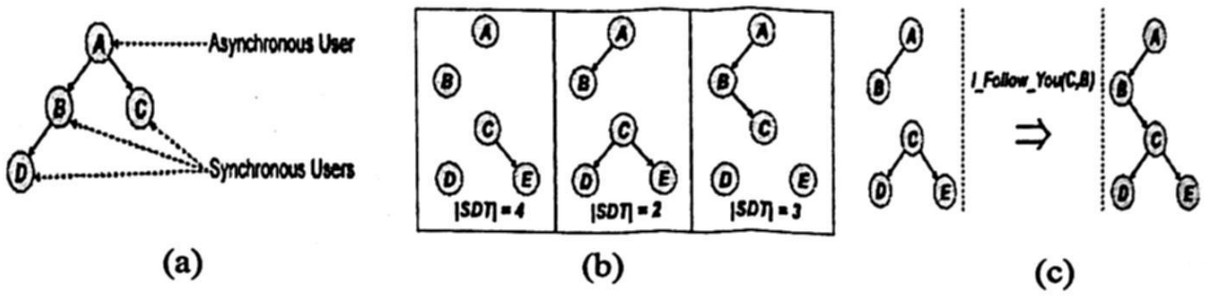

Fig. 4 (a) Basic notion of SDT, (b) SDT configuration scenarios, (c) "I_Follow_You" synchronization primitive.

Synchronization Primitives

CoLab proposes two main synchronization primitives allowing the creation of synchronization relations between users, namely: 1) "I_Follow_You" and 2) "You_Follow_Me". 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. 4c 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 |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 those of user A.

In Fig. 5a, 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" means sending

the message "message" to user *j*, and the notation "j?message" means the receival 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 the 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, 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 [1]).

1) Centralized organization, where decisions are made only at the level of the session as a whole. It is more adapted to co-browsing 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 co-browsing 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.

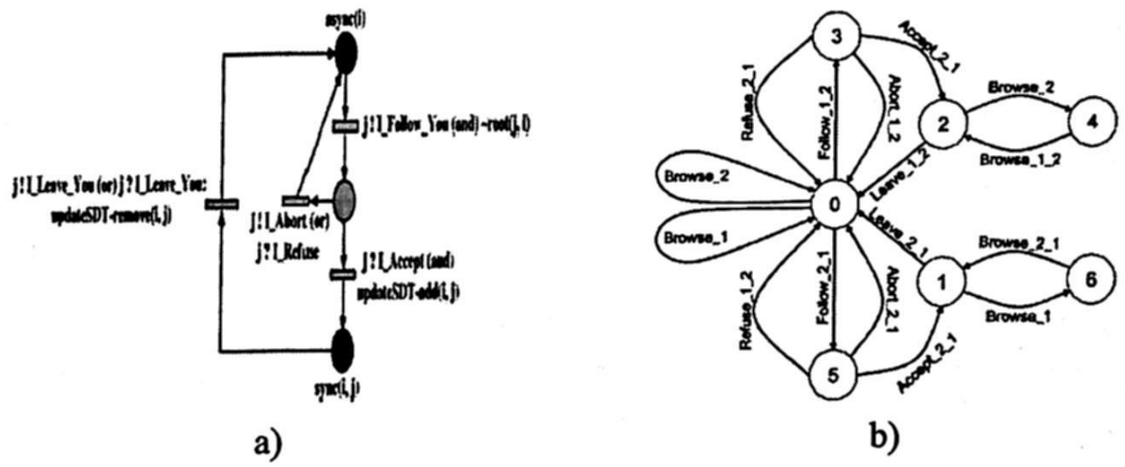

Fig. 5 (a) Basic notion of SDT, (b) SDT configuration scenarios

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 co-browsing 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" [8] to get the global reachability graph and the tool "CADP" [9] 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 [10] for details). Fig. 5b 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. 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. 5b, and we have been able to formally verify that the synchronization model is fully consistent.

5 Related Works

In [11], an adaptation from the technology of unconstrained distributed collaborative editors to develop unconstrained collaborative Web browsing is proposed. However, the effective collaboration is dependent on the awareness of context and group activity. A design of collaborative filtering service platform is described in [12].

The platform provides primitive functions for collaborative filtering that utilizes correlation of user profiles. Basic and extension functions in collaborative filtering especially in the context of distributed environment are discussed.The design of platform is fairly generalized, and it can be realized both in a centralized and peer-to-peer fashion. Furthermore, a load balancing mechanism of the platform is presented.

In [13], new methods for scape-oriented browsing, such as see-through anchors, parallel navigation, and peripheral scape presentations are presented.

A prototype system based in these methods has been designed and implemented. The system offers continuous browsing and navigation to users. A content and device management method for multiple contents browsing with multiple devices is proposed in [14]. Two concepts are introduced: (1) Content Management Description which is used to determine what content to distribute to the device; and (2) Device

Management Description which is used to determine the current status of devices available for browsing content.

This method is expected to achieve effective browsing of contents with multiple devices in users' preferred styles. In [15], a page partitioning method for collaborative browsing is proposed. This method divides a web page into multiple components and each is distributed to a different device. Furthermore, a collaborative web browsing system in which users can search and browse their target information by discussing and watching partial pages displayed on multiple devices is developed.

The closer work is proposed in [16]. Here, a service-oriented architecture for the development of advanced tools for generic service construction and composition is presented. This architecture includes technology-neutral protocols for service instantiation and management with an attempt to encourage development of corresponding tool support. Under this approach, both client side and server side are unified, and GUI services are explicitly modeled; service containers are distinguished from ordinary services that govern service management tasks.

In [17], a collaborative navigation tool called z9 is presented. This tool has as main purpose to selectively transfer anonymous navigation information among a group of users, based on identified user similarities. These similarities are the starting point for the presentation of exploration paths that potentially lead to relevant information. The mains aspects of the development of an awareness tool based on an information-oriented coordination model for synchronous collaboration sessions are proposed in [18]. This tool is supported by an adaptive layered architecture which is based on collaborative extensions of Java language, Java 3D and XML possibilities in terms of data structuring. The application field is related to the execution of a project review for the distributed collaborative design, which is applied to a spatial-domain scenario.

Finally, a Web-based multimedia learning system with automatically generated browsing structures such as hierarchical tables of contents, index and hyperlink is presented in [19]. A pilot study was conducted to evaluate the effectiveness of the system in supporting learning.

6 Conclusions

In this paper, we have defined a general-purpose proxy-based collaborative Web browsing system called CoLab, which is based on a service-oriented architecture, allowing co-browsing by means of a set of operations described as Web Services. 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 co-browsing applications.

References

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

2. G. J. Hoyos-Rivera, R. L. Gomes, and J. P. Courtiat, "A flexible architecture for collaborative browsing". In Proceedings of the 11th IEEE WetICE, Workshop Web-Based Infrastructures and Coordination Architectures Collaborative Enterprises, Pittsburgh, PA, 2002, pp. 164–169.
3. Bellwood Tom, Clément Luc, Ehnebuske David, Hately Andrew, Hondo Maryann, Husband Yin Leng, Januszewski Karsten.. UDDI Version 3.0 Published Specification. July 19, 2002.
4. Steve Vinoski. Integration with Web Services. IEEE Internet Computing. November- December 2003 pp 75-77.
5. Matthew J. Duftler, Paul Fremantle. Java APIs for WSDL (JWSDL) JSR110: JWSDL Final Release Version 1.0. IBM Corporation March 21, 2003
6. R. L. Gomes, G. J. Hoyos-Rivera, and J. P. Courtiat, "Loosely-coupled integration of CSCW systems". In Proceedings of the 5th IFIP Int. Conf. DAIS, Athens, Greece, Jun. 2005, vol. 3543, pp. 38–49.
7. G. J. Hoyos-Rivera, R. L. Gomes, J. P. Courtiat, and R. Willrich, "Collaborative Web browsing tool supporting audio/video interactive presentation". In Proceedings of the IEEE 14th Int. Workshops WetICE, Linkoping, Sweden, 2005, pp. 78–83.
8. 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.
9. Construction and Analysis of Distributed Processes. (2006). [Online]. Available: http://www.inrialpes.fr/vasy/cadp/
10. R. Milner, A Calculus of Communicating Systems, vol. 92. New York: Springer-Verlag, 1980.
11. Maria Aneiros, Vladimir Estivill-Castro. "Foundations of Unconstrained Collaborative Web Browsing with Awareness". In Proceedings of the IEEE/WIC International Conference on Web Intelligence (WI'03). IEEE Press. 2003.
12. Toshio Oka, Hiroyuki Morikawa, Tomonori Aoyama. "Vineyard: A Collaborative Filtering Service Platform in Distributed Environment". In Proceedings of the 2004 International Symposium on Applications and the Internet Workshops. IEEE Press. 2004.
13. Hiroya Tanaka, Katsumi Tanaka. "WebWalker: Scape-Oriented Web Browsing". In Proceedings of the 12th International Conference on Informatics Research for Development of Knowledge Society Infrastructure. IEEE Press. 2004.
14. Yuhei Akahoshi, Yutaka Kidawara, Katsumi Tanaka. "A Content and Device Management Method for Multiple Contents Browsing with Multiple Devices". In Proceedings of the 21st International Conference on Data Engineering. IEEE Press. 2004.
15. Takuya Maekawa, Takahiro Hara, Shojiro Nishio. "A Collaborative Web Browsing System for Multiple Mobile Users". In Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications. IEEE Press. 2006.
16. Jing-Ying Chen. "Architecting a Service-Oriented Collaborative Web". In Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services. IEEE Press. 2006
17. Andre Rodrigues da Silva, Vera Lucia Strube de Lima. "z9: An Alternative Approach to Collaborative Navigation". In Proceedings of the Fourth Latin American Web Congress. IEEE Press. 2006
18. L. M. Rodríguez Peralta, A. M. Gonçalves Silva. "A Model-based Awareness Approach for Synchronous Collaborative Sessions on the Web". In Proceedings of the Fourth Latin American Web Congress. IEEE Press. 2006.
19. Ming Lin, Jinwei Cao, Christopher B.R. Diller, Jay F. Nunamaker Jr. "Learning By Browsing: a Web-based multimedia browsing system for learning". In Proceedings of the 39th Hawaii International Conference on System Sciences. IEEE Press. 2006.