Asynchronous Interaction Method for a Remote Teleteaching Session

Lassaâd Gannoun , Philippe Dubois, Jacques Labetoulle

Institut Eurécom
2229, route des crêtes, BP. 193
06904 Sophia Antipolis, France
Voice: (+33) 93.00.26.71
Fax: (+33) 93.00.26.27
{gannoun, dubois, labetoul}@eurecom.fr

Abstract

We present a method for an asynchronous interaction during a remote teleteaching session. In a synchronous teleteaching session, the professor can view the work of students and correct them. This is provided by a shared window application. A shared window application allows collaboration-transparent, single-user application to be displayed and interacted with on multiple users' workstations. However, the functionalities of a shared window application is limited to a synchronous interaction session. In this paper, we describe an efficient method that is a step towards an asynchronous application sharing. This method provides two application services. The first service allows a student to record his X-application session and to make comments on it. The second service enables a remote professor to replay the student's X-application session. Those services help building an efficient asynchronous interaction between a professor and a student. This interaction is supported by a symmetric architecture based on WWW mechanisms, considerably eases the implementation of such an asynchronous interaction.

Keywords : X-protocol, Application Recording Service, Application Replaying Service, Asynchronous interaction, WWW mechanisms.

Introduction and Motivation

Computer Supported Collaborative Work (CSCW) is currently a very active field. Many CSCW environments require that members of a cooperating ensemble should be allowed to interact freely, i.e. without being constrained by formal procedures and established conversational conventions (Schmidt & Bannon 1992). One fundamental approach is a shared visual space, which allows the members of a cooperative ensemble each on his/her computer workstation to simultaneously share and revise information.

Shared window systems are one solution to this approach. They exploit the properties of the base window system to allow multiple users, each on their own workstation, to view and interact with a single application (Bauerfeld, 1992; Dermler, Gutekunst, Plattner, Ostrowski, Ruge & Weber, 1993).

Sharing a single user-application allows collaboration-transparent, single-user applications to be displayed and interacted with on multiple user's workstations simultaneously. The terms "collaboration-transparent" and "single-user" denote that the applications were actually constructed for a single user-environment and hence are not designed to be run in a group context (Gutekunst, Bauer, Caronni, Hasan, & Plattner 1995). A principal feature of a shared window system is that applications need not be modified to be sharable, i.e. users should be able to run applications under the shared window system without taking special preparations. The task of developing a cooperative environment is thus greatly reduced. Instead of reimplementing the numerous existing applications, the developers only have to implement a shared window system. Gutenkunst et al., underlined that the runtime behavior of applications should not be affected by the shared window system, neither in quality nor in performance. As for the latter, user interactions with applications running under a shared window system are slightly slower than those running under the base window system due to the overhead caused by the shared window system.

Standard approaches to application sharing have a major limitation, they are synchronous and thus work only when all participants are on-line at the same time. All members of a cooperative ensemble using application sharing should be synchronized and present during all the cooperating session. Application sharing would be even more effective if efficient electronic collaboration could be extended to asynchronous interaction that does not require all participants to be on-line at the same time.

This requirement is the origin of our present work at Eurecom Institute. At this institute a teleteaching platform; BETEL (Broad band Exchange over Trans-Europeen Links) was developed (Pusztaszeri, Biersack, Dubois, Gaspoz, Goud, Gros & Hubaux 1994). A teletutoring application build over this platform was demonstrated between Eurecom Institute and the Ecole Polytechnique Federale de Lausanne (EPFL). This application used a videoconferencing and shared workspace tools to allow interactions between teacher at EPFL and a group of students located at Eurecom. The teacher at one site can teach a class or supervise the students in their individual work in another site (Pusztaszeri, Biersack, Dubois, Goud, Gros & Hubaux, 1994; Gros, 1993; Marom & Gros 1993). BETEL is an effective teleteaching system because it allows students and teachers to be at different geographic locations. It is limited, however, because it does not support asynchronous teaching/learning. If the teacher is not on-line at the same time as the students, he or she can not review and correct their work.

A key aim of our work is to develop a method that allows asynchronous interaction between students and teacher as students complete practical exercises. This calls for an asynchronous application sharing service. In our current work, we want to share the Graphical User Interface (GUI) of an X-window application. Thus, this service firstly, should allow students to save their X-session activity and comment on it. Secondly, service should allow a professor to view and correct the student X-session. These capabilities should be implemented in a distributed way to allow interactive collaboration across distances.

This paper presents an efficient method which provides two application services. The first service allows a students to record their X-application sessions. The second enables a remote professor to replay this recorded X-application session in order to understand the student's work. These services are combined to allow the professor to correct the student's work. Finally we describe an architecture for an asynchronous interactive communication system using these two services. The context for this asynchronous communication is the World Wide Web(WWW).

The remainder of this paper is organized as follows: first, we introduce some fundamental features of an application sharing system. Then, we specify an approach towards an asynchronous application sharing to cope with the limitations of an application sharing system for asynchronous collaboration. Afterwards, we present the functionality of an application session replay as a fundamental step towards an asynchronous application sharing. An overview of the X window system is given, then the application recorder architecture, the storage policy, the X protocol translator and the X request scheduling policy are described. The architecture of an X replay pseudo client which provides application session replay is presented in detail. Then, an architecture for an asynchronous interaction using application session replay is exposed. Finally we provide some concluding remarks and we outline our future plans.

Fundamental Features of an Application Sharing

Several research group have attempted to design and implement application sharing over the past few years (Abdel-Wahab, H. M., & Feit 1991; Gutekunst, Bauer, Caronni, Hasan, & Plattner 1995; McFarlane, 1991; Minenko, 1995 ; Baldeschweiler, Gutekunst & Plattner 1993). They adress a number of functionalities that should be provided by an application sharing.

Workspace Management

The workspace management is responsible for visualizing cooperative work in an appropriate way. A user wants to know which items are private and are subject of cooperation. Gutekunst et al. point out that the user should be able to associate shared items to a cooperative activity and to understand the relationships among them. Under a Shared window system, a user should be able to distinguish private window from shared ones (Lauwers
& Lantz 1990) For shared windows, the user should also be able to see with whom they are shared and from which user they come from, respectively, whether or not the user may provide input to the underlying applications. This may be achieved by visual cues. Finally a shared window system should allow users to make private windows shared and vice versa.

Floor Control

Single-user applications that run under a shared window system are not aware of being run in a group context. Hence, they do not expect input from multiple users (Gutekunst et al.). "Floor Control" serves as concurrency control mechanisms that determines at any given point in time which user actions will be passed to the application and which will be filtered out. The right to generate input to the application is denoted by the "floor", the user currently allowed to do so is called the "floor holder". There are many floor control policies that comprises a set of rules

governing floor control. The default answer of passing the floor to all users is potentially confusing to all users of a cooperating work (Baldeschweiler et al.). If three people type simultaneously to a word processing program, the resulting text will be not useful. If two users push their mouse button simultaneously, the result may violate the X protocol. This policy is denoted by implicit floor passing.

In contrast to implicit floor passing, explicit floor passing is another approach to avoid inconsistency input events being sent to the application. Explicit floor passing policy allows to assign floor to a specific user. Then, users need to perform an explicit operation to request or grant the floor. This policy is a token based-model (Baldeschweiler et al.) in which the floor is passed from one user to another. The current floor holder can then interact with the application and others are excluded. Such an approach requires a user interface to allow users to request and pass the floor to each other.

Finally, there is not a preferred floor control policy. But, a convenient policy should depend on a group task, the size of the group, the politics of the group's interactions and the application itself (Gutenkunst et al.). Since any given policy will not be able to satisfy all groups in all situations, Guntenkunst et al. provide a set of floor control policies on top of a window shared system. With theses various policies, the shared window system do not enforce a specific control policy to a group of cooperating users.

User Management

Many cooperative ensembles do not occur in the context of scheduled meetings, but rather spontaneously and unplanned (Gutenkunst et al.). Users can not anticipate with whom they will be cooperating, nor which items they will be subject to the cooperative work (Lauwers & Lantz 1990). It is desirable that users be able to join and leave a cooperative activity at any time. A related aspect is admission control, it determines which users are allowed to participate in a cooperative activity as well as to join. Different policies for handling admission control are relative to the setup of a cooperating community. As cooperative ensemble intersect (Schmidt & Bannon 1992) user should also participate in a cooperating activity dynamically. Hence, dynamic user participation is a key requirement for a shared window system. Some shared window systems provide mechanisms that support dynamic user participation, e.g. X Terminal View (XTV) developed in by (Abdel-Wahab & Feit 1991). Others allow various admission policies to be realized on top of a shared window system (Gutenkunst et al.).

Towards an Approach for an Asynchronous Application Sharing

A shared window system provides a set of mechanisms to allow members of a cooperating ensemble, each on his/her computer workstation to simultaneously share and revise information relatively to an X-application. The term "simultaneously" emphasizes on the *synchronous* feature of collaboration required by a shared window system. Then, all participants should be on-line at the same time to share the work of each other relatively to a specific X-application. However, a mechanism of an *asynchronous application sharing* can be designed to cope with the limitations of a synchronous application sharing and then to allow *asynchronous* review of

work and cooperation. An asynchronous application sharing has to provide two global mechanisms:

- Asynchronous sharing of the GUI of an X-application session
- Sharing of the remote user context of an X-application

A related approach to an asynchronous sharing of the GUI of an X-application session is *application session replay* which allows a member of a cooperating community (e.g. a professor) to view different steps of a work done by another member (e.g. a student) of this cooperating community.

An application session replay is then a generic service that provides a replay of a user X-application session. This generic service should provide two application services. The first one allows users (e.g. students) to record their X-application sessions and to make comments on these recorded X-application sessions. The Second service should enable a remote user (e.g. professor) to replay these recorded X-application sessions. This replay consists of regenerating the original GUI of an X-application session as well as the different steps of this GUI evolution in time.

Sharing the remote user context of an X-application is related to an interaction between a user and a remote X-application. This X-application is then running on a context of objects relatively to a remote user. This mechanism allows a user (professor) access to a common set of objects in the remote partner (student) environment. We focus our current work on the design of an asynchronous sharing of the GUI of an X-application session. In the following we propose our approach to this mechanism based on an application session replay.

Application Session Replay

We begin by describing some of the operational characteristics of the X-Window System.

X window system

The X Window System is a window-based User Interface Management System (UIMS) providing capabilities to easily create graphical interfaces for distributed applications independently of the architecture in which the applications will be running (Sheifler & Gettys 1986). The X Window System provides network transparent access from an application to a user's display. The X Window System is both hardware and operating system independent, written application software will compile and run on any system that supports X.

Client/server model

The X-window system is a server that accepts requests to manipulate the display on the computer's console while reading input from the console's keyboard and mouse devices (Nye, 1992; Chung, Jeffay & Abdel-Wahab 1993). The client is a program displaying on the screen and taking input from the keyboard and the mouse. A client sends drawing requests and information requests to the server. The server, sends back to the client user input, reply to informa-

tion requests, and error reports (Nye, 1992). The client may run on the same machine as the server or on a different machine over the network.

Communication between the server and clients is managed by the X-protocol messages. Figure 1 presents the communication architecture between the X-server and the X-Client.

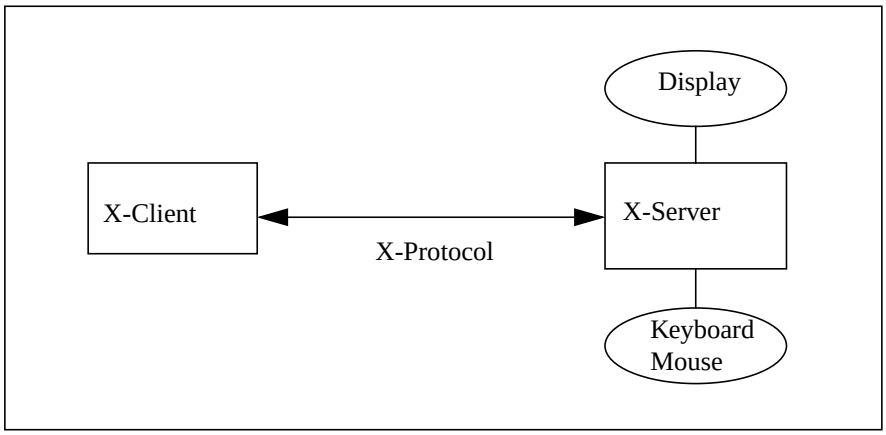

Figure 1. X-window System Architecture

There are four classes of X-protocol messages that can be transferred over the network:

- Requests: they are sent from a client to a server. There is a wide variety of requests, requests that allocate resources, requests that represent drawing and writing order on the display, e.g. for drawing a line or changing the color value in a cell in a colormap, and requests that return data needed by the client, e.g. inquiry for the current size of the window.
- Reply: they are sent from the server to the client as a response from the server to the client requests of information. These reply are useful for the client to formulate later requests.
- Events: they are sent from the server indicating to the client the user inputs from the keyboard or the mouse. The data contained in events is quite varied because it is the principal method by which clients get information.
- Errors: they are sent from the server to the client signaling that the previous request was invalid (e.g., the client specifies a window that does not exist).

For our work, the key type of messages was requests. Requests can be sent to an X-server to replay a user X-session without invoking the application. In the following section we present our recording architecture that enables to record an X-application session.

Application Recorder Architecture

The commonly used approach for sharing X-applications is to construct an X multiplexer (Baldeschweiler, Gutekunst & Plattner 1993) that intercepts the X display connection between an application and the associated X display server and then multiplexes the X protocol data

stream. In our case, we want to capture the requests sent from an X application to an X server. Then our application recorder only intercepts the connection between the application and the X server and then capture requests. We do not need to multiplex X protocol data stream over other connections. This Application Recorder provides to the user an Application ReCording Service (ARCS). This service allows the user to record his X-application session. Figure 2, shows the architecture of the Application Recorder.

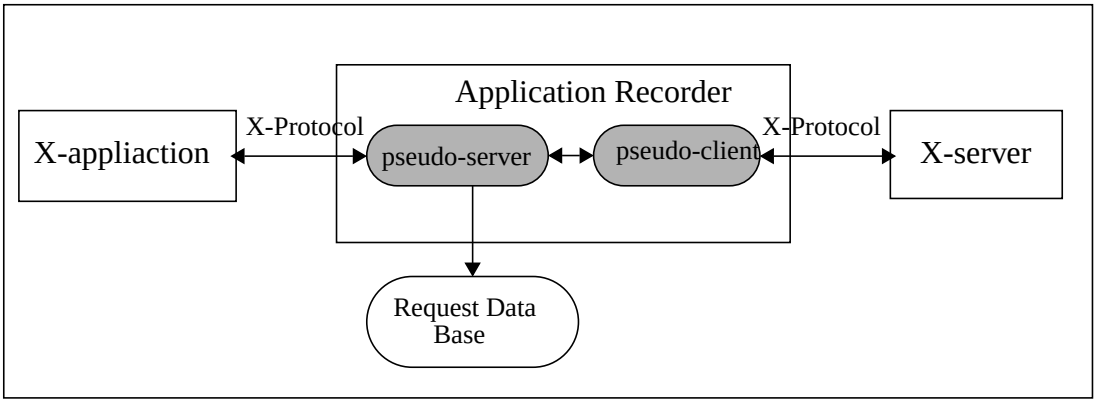

Figure 2. Application Recorder Architecture

The Application Recorder consists of two components:

- Pseudo server: This component capture requests sent from the client and store them in a data base. The *storage policy* of requests depends on the model of the X-application and will be discussed later. The pseudo server initiates the connection to the X-application and passes requests to the pseudo client. It also receives from the pseudo client events, reply and errors and passes them to the X-application. The pseudo server also filters the requests and do not store those which need replies. These requests do not affect the user display and hence can not participate to replay the application.
- Pseudo client: It plays like an ordinary client, it initiates connection to the server and passes requests to it. Once it has receives events, reply and errors from the X-server, the pseudo client passes them to the pseudo server.

The Application Recorder stores filtered requests on a data base request. These requests are stored without any modifications with respect to a storage policy.

Storage policy

The Storage Policy of requests depends on the X-application model which consists of two global models.

- A unique process model
- A multiple process model

X-application with a unique process model is associated with a unique X-client process. The request organization is then simple. Requests are stored sequentially. Requests must be timestamped to allow the replay of the session with the same perception that the original user had seen while running his application. In addition timestamps serve as a reference which allow to accelerate or to decelerate the replay of an application session. This organization allow to store the request syntax. But this is not sufficient to replay an application especially with delay parameters. In fact, the X-server analyzes the requests received from the client. This analysis is both syntactic and semantic. To preserve the semantic of the requests it is indispensable to group the requests that belong to the same context of a logical X-client demand. Requests are stored by groups. A group identifies the sequence number of requests submitted together to the X-server.

X-application with a multiple process model can generate multiple X-client processes. These processes can also generate recursively their own child processes. Replaying all these X-clients needs the storage of all their requests. These X-clients are characterized by a hierarchical relationship. However, hierarchical request storage is not necessary, if the replay algorithm sends requests to the X-server with regard to both their timestamps and their groups. Each X-client process is associated with a history request queue which stores all requests sent to the X-server by this client. Only requests that need replies are filtered and will not be stored.

To replay requests to a remote X-server, the application session replay has to accomplish two main tasks: converting and scheduling requests. The problem of converting must also be faced when sharing an X-client on multiple displays (Abdel-Wahab & Feit 1991; Abdel-Wahab & Jeffay 1992; Gutekunst & Plattner 1993). In the following we describe the X-protocol translator and the X-request scheduling.

X-Protocol Translator

As described in Minenko (1995) we denote by Secondary server, the server where the application will be replayed. The primary server is the one where the application has been run firstly. X protocol requests processed on a secondary connection server must be changed, because X servers are always different in their configurations and functionality. Menenko enumerates the principal differences between displays:

- boundaries for valid resource IDs (windows, pixmaps, graphics context etc.)
- the number of available screens
- the default depth of all screens
- the number and default parameters of visuals of each screen
- supported pixmaps

The first step for translating requests is to handle new parameters (or general information) about the secondary server (e.g. root windows, default colormaps and visuals), because these parameters allow to convert stored requests that contain the same parameters with respect to the primary server.

The main items subject to conversion are resource identifiers, and pixel values. In Abdel-Wahab & Feit (1991) and Abdel-Wahab & Jeffay (1992) these problems are described in more detail. All resource identifiers in requests are to be mapped to new values in order to fit into the destination boundaries. These boundaries are handled from the new connection with the secondary server. Mapping is done using general resource information and private resource information (between the X client and the primary server e.g. resource-id-base and resource-id-mask) of both the primary server and the secondary server. This needs that we should also store general resource information and private resource information between the primary server and all X-clients of an X-application.

Besides, resource identifiers, requests refer (directly or indirectly) to other graphic parameters defined in the X protocol such as pixel values, font names, images, etc. (Minenko 95). Graphic parameters may vary in sessions with different servers and even in different sessions with the same server. Then, unlike the identifier mapping, the graphic parameter mapping do not usually produce the authentic representation of the GUI on the secondary server. This also depends on the graphical mapping algorithms usually applied to provide the best representation of the GUI on the secondary server. The quality of these applied mapping algorithms determines how frequently the “best representation” of the GUI turns into the “equal representation” (Minenko 95).

Once the all stored requests are translated to the convenient ones accepted by the secondary server, a scheduling policy is applied for their replay.

X-Request Scheduling policy

This policy replays requests in an order determined by their timestamps and their groups. Even if there exists some relationships between X-client processes, requests that belong to an X-client are stored alone in a history request queue. Precedences between requests are solved by timestamps. A simple algorithm that replays efficiently an X-application is similar to an optimistic distributed simulation algorithm. Each history queue associated with an X-client is considered as a channel. These history queues are channel inputs to the scheduler. A global variable GroupReq is defined as the next request group order to be loaded. The algorithm selects the request with the low GroupReq among all requests at the head of each request queue. Then, a specific queue is selected and all requests that belong to the same group are sent to the X-server. To select the next request group the scheduler increments the GroupReq Variable by 1.

Each history request queue is associated with a particular connection with a server. A group of requests selected is sent to a server connection with respect to its request history queue. Request timestamps serve to compute the delay before sending requests. This delay can be modified to accelerate or to decelerate the replay of an X-application.

Architecture of an X-Replay Pseudo Client

An X-replay pseudo client is a virtual client that provides to the user the Application RePlaying Service (ARPS). This service allows the user to replay a stored X-application session. The pseudo client is substituted for the real X-application and sends convenient stored requests to the X-server. Its main tasks are: loading and scheduling requests, translating loaded requests

and making requests, and managing X-server connections. Figure 3, shows the architecture of an X-replay pseudo client.

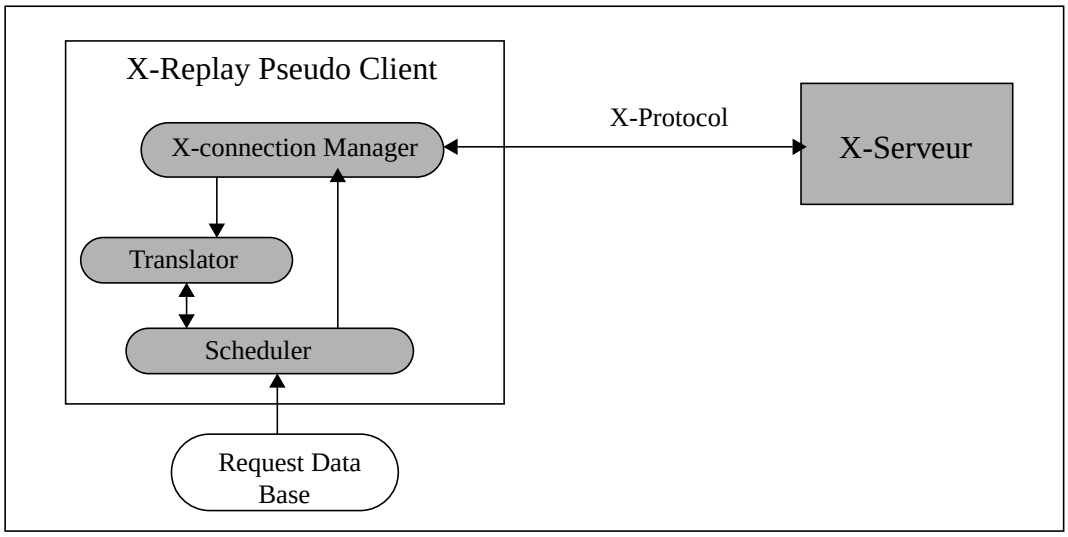

Figure 3. Architecture of an X-replay Pseudo Client

The scheduler module loads the requests, the private informations between the primary sever connections and X-clients, and the general informations about the primary server. This module uses the scheduling policy to schedule groups of requests. To convert a first group of requests from a given history queue, this module should initiate a first connection to a server. The X-connection manager gives the translator the real time information about the current connection being initiated. This allows translation of requests by using the stored information about primary server. If the deadline of sending group of requests occurs, the loading and scheduling module passes the current group of requests to the X-connection manager that sends them. Response of the X-server such as fake events and errors can be decoded and displayed.

Replaying an X-application session is indispensable for an efficient asynchronous viewing and interaction. In the following we describe how ARCS and ARPS are used to build an efficient asynchronous interaction.

Asynchronous Interaction Using X-Application Session Replay

Efficient asynchronous interaction between a professor and a student uses the services provided by the X-application session replay. This interaction is supported by the WWW mechanisms.

Why World Wide Webs? WWW offers two strong mechanisms; HyperText Markup Language (HTML) and the HyperText Transport Protocol(HTTP) (Frivold, Lang & Fong 1994). Textual comments and questions are structured in HTML documents. These HTML documents include

also starting points to replay an X-application session. Links to other documents can be added to better understand comments. We present the following scenario to explain how HTML documents are used in such an interaction.

Typical usage scenario

When facing a problem the student records his or her X-application session. Students begin by making textual comments to introduce their work and the first problem encountered. These textual informations are stored as an HTML document. During the X-session the student stops running the application and makes comments about the problem encountered. Figure 4 shows the first step of an interaction made by a student to raise questions about the problems encountered during a session of work.

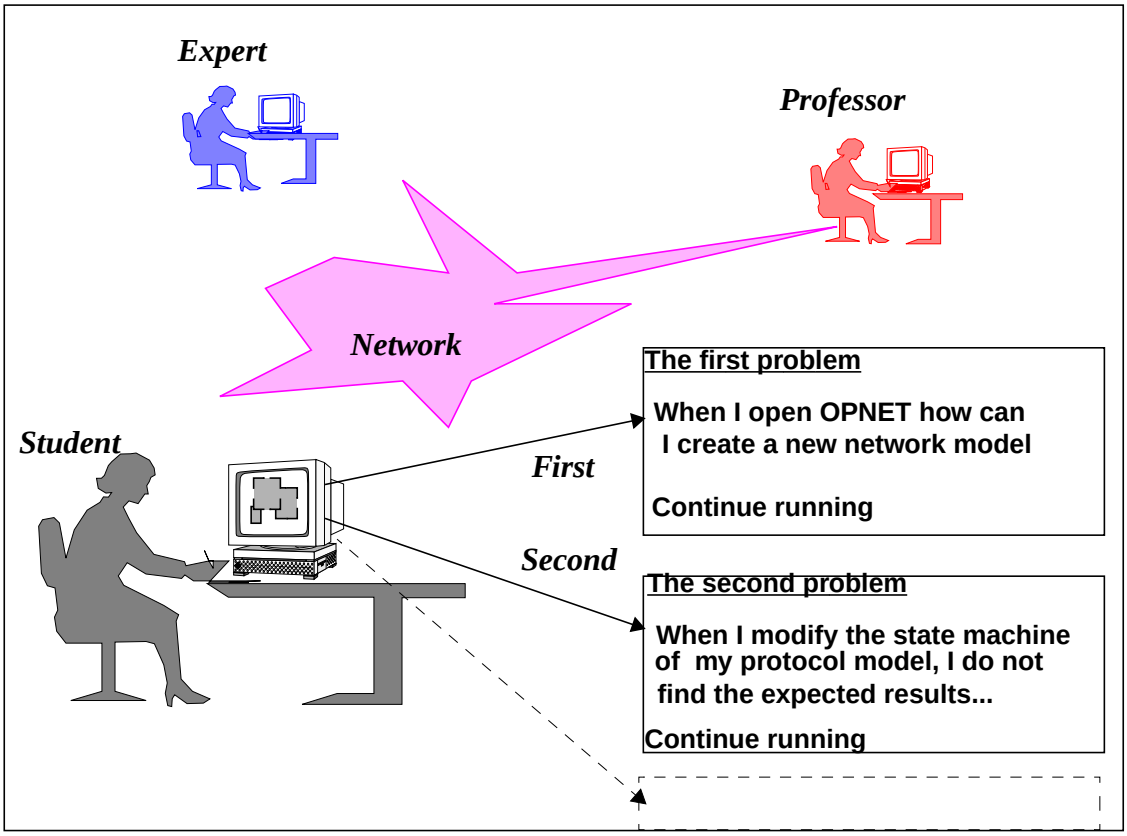

Figure 4. Student Questions to the Professor

On the other side, the professor loads an HTML document that begins with an introduction to the work and the problems encountered by the student. After reading the problem introduction the professor press a starting bottom to begin the replay of the student's X-session. Replaying is interrupted by halt points that displays the student's comments and questions. Figure 5 presents the second step of an interaction which is the correction of student's work and the making of indications to solve encountered problems.

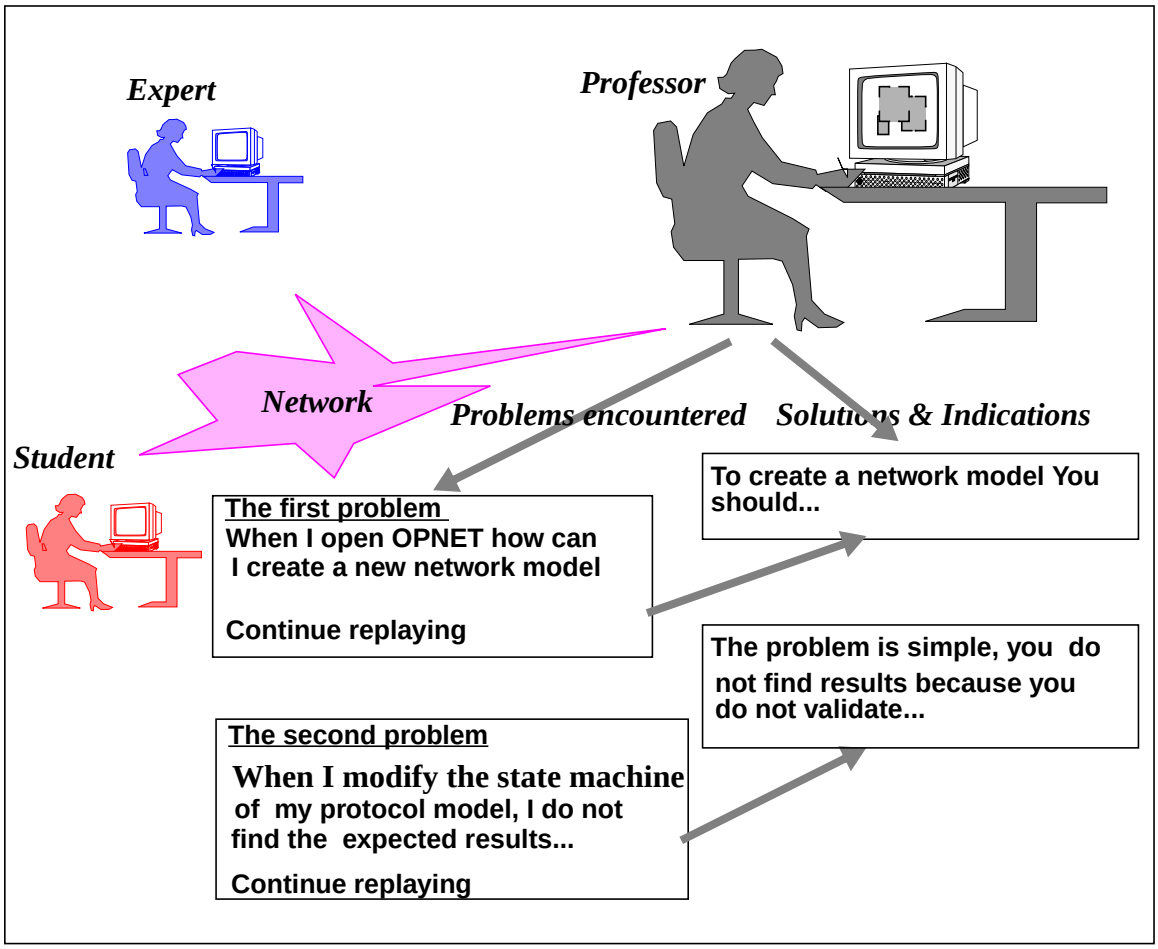

Figure 5. Professor Responses to Student

The professor, reads the student questions and for each question the professor makes the corresponding comments and indications that help the student to find the solutions of the current problem he or she encountered.

Global interaction architecture using WWW

In the following section we present the interaction architecture that uses WWW. We denote by X- session profile both informations stored about the X-session and the textual comments and questions. Figure 6 shows the interaction architecture through which the student communicates with the professor. Three steps are distinguished:

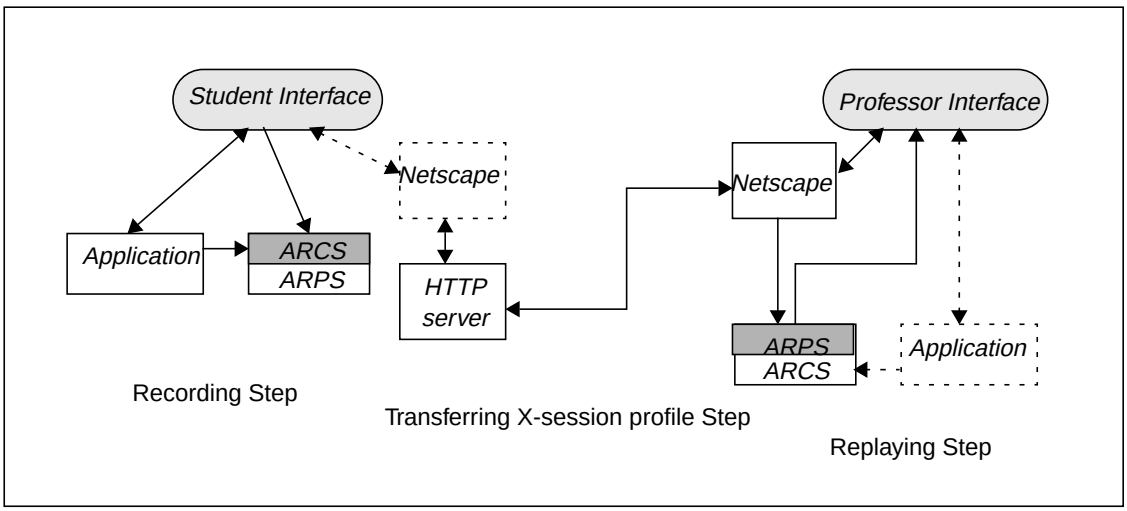

Figure 6. Interaction Architecture Using WWW

Recording step: during this step the student's X-session is stored. Student's comments and questions are also stored as HTML documents. Relationships between comments and X-session portions are established. All these informations constitute an X-session profile.

Transferring X-session profile step: This step is done later by the professor to respond to the student's invitation. This invitation is simply done by a message mail in which the student gives the professor the Uniform Resource Locator(URL) address of the X-session profile. With this address the professor can load the X-session profile and initiate the replay of the student's session.

Replaying step: in this step the professor can replay the student's X-session. The replaying is interrupted by halt points. These points allow to display the student's comments and questions. After reading these comments a go-on bottom enables the professor to continue replaying the X-session. In order to correct the student's work the professor have to make indications to the student. Indications are either textual or explicit X-session demonstrations that shows the student how he should proceed to solve the problems encountered. This needs switching from a replaying step to a recording one. To build an efficient X-session profile response, we can have replaying and recording steps in parallel. This allows the professor to make indications on the student's X-session.

To get the feedback, the student should load the professor's X-session profile. The reference address of this X-session profile is given to the student by a professor's mail message. To view the professor's indications the student should replay the X-session being transferred. Hence our interaction between the student and the professor is based on a symmetric architecture. This exploits the client/server paradigm of WWW mechanisms.

A major drawback of this symmetric architecture is that it generates an overhead of information on the professor's machine. All students responses are stored on this machine. However distributing responses to corresponding students leads to allow the professor to write X-session

profiles on remote machines. An asymetric architecture can provide distribution of X-session profiles. This needs to build a daemon server on each student machine. This server allows the professor to send X-session profiles to the corresponding student.

Conclusion

The goal of our work was to design a method for an asynchronous interaction during a teleteaching session. A critical analysis of a shared window application shows that this approach has a major limitation to allow asynchronous sharing of the GUI of an X-application. To cope with these limitations we have specified an asynchronous application sharing as an approach for an asynchronous sharing and revise of information. A fundamental step towards this approach is application session replay which enables an asynchronous sharing of the GUI of an X-application.

Two complementary services are designed and developped to allow application session replay. The first, is the ARCS which allows a student to record his X-application session and to add comments along all his/her session. The second is the ARPS enabling a remote professor to replay the student's X-session and to understand the student's work. These services are combined together to correct the student's work. Corrections are either textual indications or X-application demonstration that helps the student to improve him/her self.

These two complementary services are indispensable for an asynchronous sharing of the GUI of an X-application. But they are not sufficient to enable an asynchronous interactionbetween the student and the professor. Then, we propose to integrate these services with an interactive communication architecture. Worl Wide Webs mechanisms are effecient and well suited for an asynchronous cooperation and constitute the communication based architecture that we have adopted. Interaction between the professor and all students is supported by a symmetric architecture based on WWW mechanisms.

In future work, we hope to develop a system that allows a selective replay of an X-session. This would allow the professor to replay a specific portion of a recorded X-session. This problem is similar to a dynamic sharing problem which attempts to allow a late-comer to participate in a conference session (Chung et al., 1993). It would also be interesting to add the ability to switch from the replay of a specific portion to another one of a recorded X-application session. Finally, we should allow a professor to go back and replay a previous portion of an X-session.

References

Abdel-Wahab, H. M., & Feit, M. A. (1991). XTV: A framework for sharing X-window clients in remote synchronous collaboration. *Proceedings of the IEEE Conference on Communica-*

tions Software : Communications for Distributed Applications and Systems 1991 (pp. 159-167). Chapel Hill, North Carolina.

Abdel-Wahab, H. M., & Jeffay, K. (1992). Issues, problems and solutions in sharing X-clients on multiple displays. *Technical report TR92-042*. University of North Carolina. Chapell Hill.

Baldeschweiler, J., Gutekunst, T., & Plattner, B. (1993). A survey of X protocol multiplexor. *ACM SIGCOM* 1993, (pp 16-24).

Bauerfeld, W. (1992). RACE-Project CIO (R2060): Coordination, implementation and operation of multimedia tele-services on top of common communication platform. *Pro ceedings, of the International Workshop on Advanced Communications and Applications for High Speed Networks: IWACA'92* (pp. 401 - 405) Munchen.

Chung, G., Jeffay, K., & Abdel-Wahab, H. M. (1993). Accommodating late-comers in shared window system. *IEEE Computer, 26(1)* pp. 72-74.

Dermler, G., Gutekunst, T., Plattner, B., Ostrowski, E., Ruge, F., & Weber, M. (1993). Constructing a distributed multimedia joint viewing and tele-operation service for hetero geneous workstation environments. *Proceedings, of the fourth IEEE Workshop on Future Trends of Distributed Computing Systems* (pp. 8 - 15). Lisbon.

Frivold, T. J., Lang, R. E., & Fong, M. W. (1994). Extending WWW for synchronous col laboration. *Electronic Proceedings of the Second World Wide Web Conference' 94: Mosaic and the Web* [On-line]. Available: http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/CSCW/frivold/frivold.html.

Gros, P., (1993). Remote teaching application: Dialog states and transitions, connection. *Technical report 1-EURE-008*. Institut Eurecom, France, 1993.

Gutekunst, T., & Plattner, B. (1993). Sharing multimedia applications among heterogeneous workstation. *Proceedings, Second International Conference on Broad Band Islands* 1993 (pp. 173 - 191). Athens.

Gutekunst, T., Bauer, D., Caronni, G., Hasan, & Plattner, B. (1995, February). A distributed and policy-free general-purpose shared window system. *IEEE/ACM Transactions on Networking*.

Lauwers, J. C., Lantz, K. A., (1990). Collaboration awareness in support of collaboration transparency: requirements of the next generation of shared window systems. *Proceedings of the ACM CHI'90 Conference (Human Factors in Computing Systems),* pp. 303 -311. Seattle.

Marom, R., & Gros, P. (1993). Remote teaching application ergonomics/U.I specifica tions. *Technical Report 1-EURE-006.* Institut Eurecom, France.

McFarlane, G. (1991). Xmux-a System for computer supported collaborative work. *Pro ceedings, of the 1st Australian Multi-Media Communications, Applications and Technology Workshop,* (pp. 12 - 28). Sydney.

Minenko, W. (1995). The application sharing technology. *The X advisor June 1995 -vol 1 No 1.* [On-line]. Available: http://landru.unx.com/DD/advisor/. Discovery Publishing Group.

Nye, A. (1992). X Protocol Reference Manual. Sebastopol: O'Reilly & Associates.

Pusztaszeri, Y. H., Biersack, E., Dubois, Ph., Gaspoz, J. P., Goud, M., Gros, P., & Hubaux, J.-P. (1994). Multimedia teletutoring over a trans-european ATM network. *2nd IWACA Conference*. Heidelberg, Sept 1994.

Pusztaszeri, Y. H., Biersack, E., Dubois, Ph., Goud, M., Gros, P., & Hubaux, J. P. (1994). Teletutoring over BETEL network. *15th speedup workshop*. Lugano.

Sheifler, R. W., & Gettys, J., (1986). The X window System. *ACM Trans. Comput. Graphics*, pp. 79- 109.

Schmidt, K., & Bannon, L., (1992). Taking CSCW seriously. *Computer Supported Coopeative Work (CSCW),* 1(1-2), pp. 7-40. Kluwer Academic Publishers.

Lassaâd Gannoun is a Ph.D. student at the Corporate Communication departement. Eurecom Institute 2229, route des crêtes BP: 193, 06904 Sophia Antipolis, France. Voice: (+33) 93.00.26.71, Fax: (+33) 93.00.26.27, E-mail: gannoun@eurecom.fr

Philippe Dubois is an Engeneer Researcher at the Corporate Communication departement. Eurecom Institute, 2229, route des crêtes BP: 193, 06904 Sophia Antipolis, France. Voice: (+33) 93.00.26.44, Fax: (+33) 93.00.26.27, E-mail: dubois@eurecom.fr

Jacques Labetoulle is Professor and Director of the Corporate Communication departement. Eurecom Institute, 2229, route des crêtes BP: 193, 06904 Sophia Antipolis, France. Voice: (+33) 93.00.26.10, Fax: (+33) 93.00.26.27, E-mail: labetoul@eurecom.fr