Transparent Latecomer Support for Web-Based Collaborative Learning Environments

Torsten Illmann, Rene Thol, Michael Weber

Department of Multimedia Computing, University of Ulm, Germany

torsten.illmann@informatik.uni-ulm.de, rene.thol@gmx.de, weber@informatik.uni-ulm.de

ABSTRACT

In this paper we examine the problems of synchronous collaboration of users in web-based learning environments. It is a strong challenge to develop efficient synchronous groupware systems which provide transparent collaboration of existing applications whereas participants may start at different points in time. Existing collaboration systems either provide transparency or the accommodation of latecomers. We developed a transparent support for accommodating latecomers in collaborative environments which may be integrated in any Java-based system on the web.

Keywords

CSCW, CSCL, Synchronous Groupware, Latecomer Support, State Migration, Serializable Java User Interfaces

INTRODUCTION

Virtual and especially web-based learning is popular. Within the project "Docs 'n Drugs - The Virtual Policlinic" (Illmann, et al., 2000) we are developing a web-based and case-oriented training systems for medical students. Students learn to come to case-based decisions by answering questions or interpreting/examining findings displayed as multimedia elements. Since the system is already embedded in the curriculum of medical students at the University of Ulm, it is often used and many cases are currently in development. Web-based applications realized as Java applets enable to process and create cases.

A big challenge of such systems is to support shared learning and authoring in location-independent groups. One distinguishes between synchronous and asynchronous collaborative learning.

When evaluating existent synchronous collaboration frameworks for our system, we noticed a system called JASMINE (Saddik, et al., 2000) which provides transparent synchronous collaboration of Java applets and applications. Unfortunately it does not support latecomers. That means that all participants of a collaborative session must start the program at exactly the same time.

In this paper we present a transparent support for latecomers of UI-based applications in Java. We outline how synchronous collaboration in Java can be achieved and describe how transparent latecoming can be supported. We integrated our implementation in the JASMINE system.

JAVA AND COLLABORATION

To realize synchronous collaboration in Java, applications or applets may be used. Since applets are special-designed Java applications for the WWW, they are a good choice for implementing collaborative applications in Java. Applets reside on a web server. On request, they are transferred to and executed on the client computer with the permission to communicate with the web server host. These are ideal conditions for implementing a collaboration framework in Java with applets.

A sophisticated collaboration framework should be able to support collaboration for any existing applet. If there are fix interfaces to meet, applet programmers have to know the interfaces and the applets do not run without the collaboration framework any more. To achieve this goal of transparency, the framework must integrate collaboration in the applet without the applet's knowledge. Its main task is to transparently forward UI events to all other participants within the current session. Forwarding events happens in three steps: Catching events, distributing them and triggering them to the user interface on the other location(s).

1. To catch all UI events, one may traverse the total UI component tree (starting from the root pane) and subscribe to each component for all possible events. This mechanism is quite time and data expensive. Another possibility is the registration of a general callback at the default UI toolkit of Java for all events using the method `Toolkit.getDefaultToolkit().addAWTEventListener`.

2. Events which have been caught have to be distributed to all other participants. This includes the transformation of them into a serializable structure which contains an index of the component they have been released on. The distribution may be performed by a central dispatch server where participants of this session are registered or by using a multicast-capable publish/subscribe communication infrastructure.

3. When a remote event is received, it has to be triggered to the corresponding component on which it originally has been released. The index identifies the component and the event is triggered by the `dispatchEvent` method of the component.

LATECOMING

Based on the ideas described above, a transparent support for latecomers has been developed. All applications that are commonly used by several users must be grouped to a collaborative session. If a latecomer is willing to accommodate collaborative applications that are already running at one or several other locations, two tasks have to be ensured by the collaboration framework:

- The state of the collaborative applications has to be transferred to the latecomer.

- During the transmission of the state (which lasts some time) none of the collaborative applications may change their state .

In order to transfer the application's state, one has to choose one collaborative applications that overtakes this task. To avoid problems to capture the program counter and local stack frames (Truyen, 2000), we must ensure that the control flow of the chosen application resides in the main event loop. Furthermore, we suppose that the application consists of only one running thread and has no open connections to resources such as files or databases. Taking these requirements into account, we may serialize the whole applications with starting from the root pane using the Java serialization mechanism (Sun, 2001). This mechanism requires all objects to be serializable. Fortunately, standard Java classes (except above mentioned exclusions) and elements of the Java UI framework (Swing and AWT) are already serializable. Unfortunately, event listeners which are subscribed to UI components are not serializable and therefore get lost or produce undesired exceptions during transmission. The only way to fix that problem transparently is to patch the base interface of all event listeners, the `java.util.EventListener` by extending the `java.io.Serializable` interface. Hence, all other event listeners (standard and custom ones) automatically get serializable by inheritance. Using that small patch , the serialized application can be transmitted to the latecomer's location. There, the application is deserialized, prepared for collaborative use and shown to the latecoming user.

The second task for latecomer support is to lock all collaborative applications simultaneously in order to avoid state-changing events during transmission. Since a simultaneous invocation of these operations without a synchronized common physical time among all participants is not possible, all applications are requested to disable asynchronously. User events that occur before all applications have acknowledged to be blocked are buffered in a message queue and have to be sent to all applications (except the initiating one) after the latecoming process.

FURTHER INFORMATION

To get more detailed information of this work a longer version of this paper may be accessed at http://www.docs-n-drugs.de. This version includes a detailed introduction, a quantitative analysis, related work and a discussion of problems and limitations of this approach.

REFERENCES

Truyen, E., et al. (2000) Portable Support for Transparent Thread Migration in Java, *Proceedings of ASAMA'2000*, (Zuerich, Germany, 2000), Springer, 29-43.

El Saddik, A., Shirmohammadi, S., Georganas, N., Steinmetz, R. (2000) JASMINE: Java Application Sharing in Multiuser INteractive Environments. *Proceedings of IDMS '2000* (Enschede, Netherlands, 2000), Springer, 214-226.

Illmann, T., Weber, M., Martens, A., Seitz, A. (2000) A Pattern-Oriented Design of a Web-Based and Case-Oriented Multimedia Training System in Medicine. *Proceedings of IDPT '2000* (Dallas, USA, June 2000), Social Science Computing Review.

Sun Microsystems Inc. (2001) Java Object Serialization Specification, ftp://ftp.java.sun.com/docs/j2se1.3/serial-spec.pdf, visited 10.10.01