In Proc. of the 12th International Symposium on
Intelligent Multimedia and Distance Education
(ISIMADE'99), Baden-Baden, Germany, pages 203-209,
August 1999. ISBN 0-921836-80-5

Collaborative Working with Stand-Alone Applets

Abdulmotaleb El Saddik[1], Oguzhan Karaduman[1], Stephan Fischer[1] and Ralf Steinmetz[1,2]

[1] Industrial Process and System Communications
Dept. of Electrical Eng. & Information Technology
Darmstadt University of Technology

[2] GMD IPSI
German National Research Center
for Information Technology

{abed, oguzhan, fisch, rst }@kom.tu-darmstadt.de

ABSTRACT

In this paper, we describe an approach to support col[1]laborative working with unknown applets (source code is not available). The general idea of the our system is that user events occurring through the interaction with the GUI of an applet can be caught, distributed, and reconstructed, hence the system allows for Java applets to be shared transparently. Our approach differs from other collaborative systems in modifying their source-code. The architecture described here allows to make a lot of already existing applets collaborative, which have been developed as single user applications with no collaboration in mind.

1. Introduction

The simplicity of access to a variety of information stored on remote locations led to the fact that the World Wide Web has gained popularity over the last decade. In this context Computer Supported Collaborative Learning (CSCL) is becoming more and more important. Collaborative systems allow users to view and interact with a distributed application during a session. The use of collaborative systems increases in research and business as well as in education. A problem of many cooperative applications is their platform dependence leading to the fact that users communicating in heterogeneous environments are restricted in their choice of a cooperative application. For example a user might choose a UNIX-workstation, another might prefer Windows 95/98/NT or a Mac-intosh. The introduction of the platform-independent programming language Java made it possible to overcome these problems. Diverse approaches were used to develop Java-based collaborative systems. Almost every system described in the literature requires the use of an Application Programming Interface (API). Others are trying to replace some Java-components with self-defined collaborative components in a transparent manner.

The approach presented in this paper differs from other approaches, in the way that we do neither propose a new API for developing collaborative systems nor try to replace components at run time. As a matter of fact a great variety of well-designed applets exist distributed in the World Wide Web which were developed to run stand-alone. It would not be acceptable for many users to re-implement or change these programs to make them work in a collaborative way. The particularity of our approach is that we propose to use JavaBeans [1] capabilities of applets in a way that stand-alone applets can be used in a collaborative way. Our approach does not change the source code of an applet. The System we developed is called KOM Collaborative Applets Environment (KCAE). The principal idea of the KCAE-system is that user events occurring through the interaction with the GUI of an applet can be caught, distributed, and reconstructed, hence allowing for Java applets to be shared transparently. This form of collaboration which is supported as long as a learning-session takes part, enables users to interact in real-time, working remote as a team without caring about low-level issues such as networking details.

Figure 1 shows the overall approach of our KCAE-system which can be described as follows: a Collaboration client (KCAE-Client) instantiates applets or

[1] In Proc. Of Intelligent Multimedia and Distance Education (ISIMADE '99), August Baden-Baden, Germany 1999.

applications which are developed as stand-alone applications. These applets or applications are then used collaboratively with the aid of the KCAE-Client. The KCAE-Client can be seen as a component adapter. Every event occurring at the graphical user interface of the application is sent to the adapter, which then sends the events to the collaboration server (KCAE-Server). When the KCAE-server receives an event, it multicasts it to all other KCAE-clients in the session.

The rest of the paper is organized as follows. Section 2 describes the system design. The description of the architecture (Section 3) is followed by the description of the collaborative use of source code unavailable applets in Section 4. Section 5 discusses related work and Section 6 concludes the paper and gives an outlook.

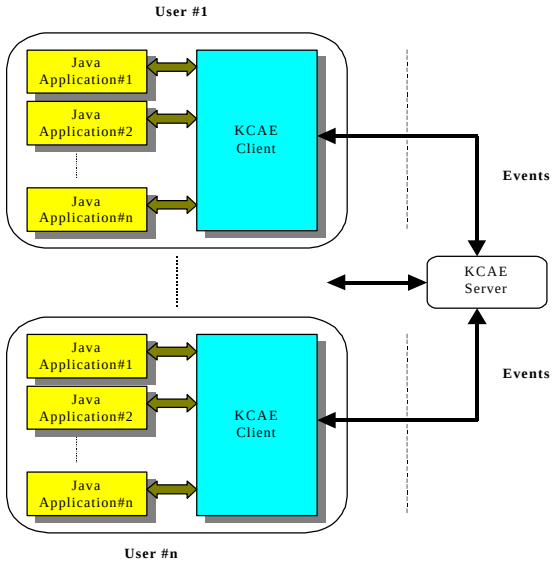


Fig1: Overall System Architecture of KCAE

2. System Design

available applets collaborative, which are developed as single user applications with no collaboration in mind, without any change or modification of their respective source code.

2.1 Requierements

Before describing the architecture of the system, we first present a list of requirements which describe our objective in more detail:

Applications using the standard Java-Core API should be supported, that is no change in the API should be necessary.

No source code is required to share an applet. Both AWT and Swing components should be supported. A solution restricted to only one kind of graphical user interface is not acceptable.

The system should permit unanticipated collaboration. Furthermore, a person who is interested in more than one application running at the same time, should have the opportunity to take part in more than one application simultaneously.

be consumed.

2.2 Design Considerations

The delegation event model of JDK1.2 provides a standard mechanism for a source component to generate an event and send it to a set of listeners. Furthermore, the event model even allows to send the event to an adapter, which then works as an event listener for the source and as a source for the listener. Because the handling of events is the crucial task in developing an application, this enhancement made the development of applets much more flexible. Another important enhancement of Java is the introduction of the JavaBeans Technology and of the Remote Method Invocation (RMI) which makes it possible to create distributed Java-to-Java applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly running on different hosts.

There are two main approaches for the design of collaborative applications [15]. One possible approach implies a centralized architecture, where a single instance of an application is run on a host machine, usually the server machine, and its graphical output is distributed to all participants. This approach which is illustrated in Figure 2 consumes substantial network bandwidth (illustrated through bold arrows in Figure 2), even if the graphical data is transmitted in a compressed way [14].

Another approach is the replicated architecture which allows the application to be downloaded to each participant and to run locally. Consequently, the bandwidth required for the collaboration is substantially less than using a centralized architecture [14]. Only input resulting from the interaction with the graphical user interface is distributed, the graphical output is generated locally for each participant.

The bandwidth savings become apparent when one considers that the centralized approach also must

receive input from each user, but uses a considerable amount of bandwidth to distribute the graphical information.

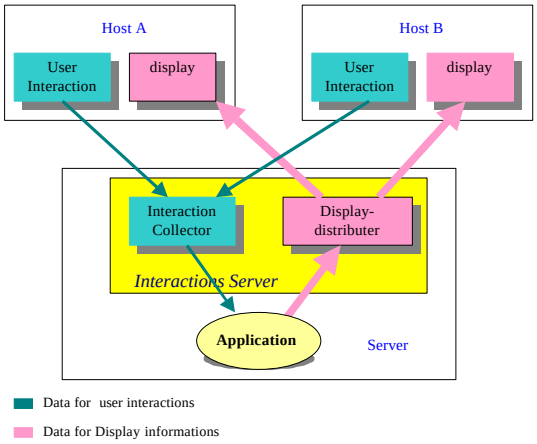


Fig2: Interaction by transmission of graphical data

Our system is based on the replicated architecture presented in Figure 3 in which an instance of each appli
only the interaction of each user with the system is distributed through events to all the participants passed by the server.

It should be noted that our approach relies on the particular properties of Java. The Java Virtual Machine provided in each browser available nowadays offers a homogeneous application environment across different platforms. All participants in a collaborative session have access to the applets which are downloaded and

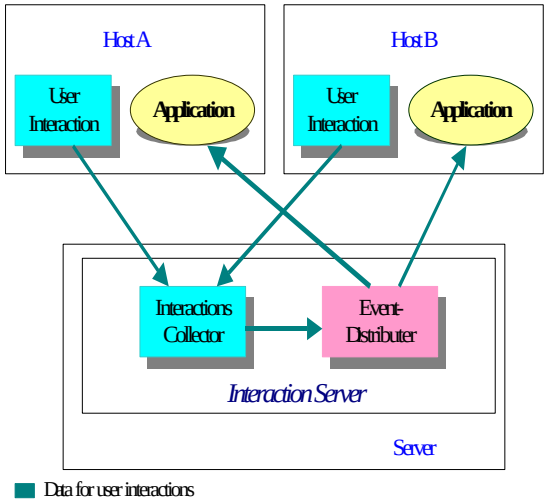
of using Swing components the use of a Java-Plugin to

solution.

3. System Architecture

In this section, we describe our implementation of a collaborative system according to the requirements and design consideration described above. As we are concerned with Java applets it should be mentioned that we presume the availability of a web-server. Therefore it is the part of the system to start with. It should also be noted that there are two types of servers running on the central machine. One is the Web-Server which sends the HTML documents and the applets to the requesting clients, the second one is a Java-RMI server responsible for multicasting all events sent from one client to all other clients.

When the user loads an HTML document that contains a reference to a KCAE-Client, the browser loads the



applet and executes it. When the applet is started the

Fig3: The replicated distribution approach

user requests to join a specified session. If the session does not exist, a new one is created. If no other participants are in the session the unique person in the session can interact with the applet s/he chose from a list of all available applications and explore it as if s/he were not be part of a collaborative environment. This is done because applications are designed and implemented as collaboration-unaware applications. Thus the KCAE-client sends the received events to the KCAE-Server, which finds out that there are no other participants in the session and ignores the events.

Java Applications/Applets

It should be noted that Java applications/applets instantiated by our KCAE-Client are not part of KCAE. They are furthermore collaboration-unaware applications developed using the standard Java technology. As mentioned earlier in this paper, our system supports both AWT- and Swing-based applications. These applications are loaded dynamically, after a client joined a session. In this manner all possible applications/ applets a user can invoke can be stored in a configuration file by the KCAE-Client. Participants can invoke one or more of these applications in a session if desired.

3.1 KCAE-Server

The KCAE-Server is a Java based application. It is entirely written in Java and can therefore be run

independently from the underlying platform. The server works as an event multicaster. Each remote client in the collaboration session registers itself with a unique name at the server.

The communication between KCAE-clients and KCAE-server is based on Java RMI and bi-directional. With the help of the RMI-Callback mechanism, the server communicates directly with the registered clients. The server dispatches the external events coming from registered clients in the session and multicasts them. An event received by the server is multicasted to all registered clients except of the client which fired the event. Information about available applications and applets, which can be used in a collaborative way, are read from the configuration file by the server and sent to participants who just joined, so that the participants can load and run these applications and applets. The configuration file, which is organized as a properties file, contains the names of the applications/applets and the full names of their main class. The entries should have the following syntax:

application.[n].name = [name]

application.[n].class = [class]

where:

n: number of the application in the list.

name: a suitable name for the application.

class: full name of the main class.

An example is in Figure 4 illustrated:

```
#Application entry
application.1.name=myTestApplication
application.1.class=kom.develop.apps.MyApp
# Applet entry
applet.1.name=myTestApplet
applet.1.class=kom.develop.applets.TestApplet
```

Fig4: Excerpt of a configuration file

3.1 KCAE-Client:

The KCAE-client is a Java applet which consists of the following components:

- Collaboration Manager
- Component Adapter
- Listener Adapter
- Event Adapter

Collaboration Manager

The Collaboration Manager is the main component on the client side and provides the user with a graphical interface offering the following options: Joining the session, starting applications/applets and chatting with all other participants. The collaboration manager dispatches external events coming from the collaboration server and forwards them to the component adapter.

Components Adapter

The Component Adapter maintains a list of the GUI-components of all applications and applets. This list is created with the help of the java.awt.Container-class, which allows to get references of all applet components. With the help of the main window of an application, a list of the GUI components in this application can be created directly. For that reason, the main window of an application loaded by the Collaboration Manager is registered by the Component Adapter. However, Java applets do not use stand alone windows. They are an extension of the class java.awt.Panel and can thus be easily located in a window. The window containing the applet can then be registered as main window by the Component Adapter. As an example syntax of the registration by the Component Adapter is shown in Figure 5. After the registration the list of all the Swing and/or AWT-components within the applet is created directly. This task is done by each client in such an order, that the components have the same reference by all clients. The references are used to point the components, which are the source of the events sent to the Server and after that multicasted to all clients. With the help of references, the source of the incoming events are located and the event is reconstructed on each client, as if it occurred locally.

```
.....
Class cl = Class.forName(className);
// If it is an applet, instantiate and locate
// it in a Frame
myApplet = (Applet)cl.newInstance();
myApplet.init();


myWindow.add("Center", myApplet);
// Otherwise instantiate it
myWindow = (Window)cl.newInstance();
// Register this Frame as main Frame
```

```
// by Components Adapter
ComponentsAdapter.addContainer(myWindow);
....
```

Fig5: Excerpt of the instantiation method

Listener Adapter

The Listener Adapter implements AWT listeners, which listen to Mouse- and KeyEvent for all AWT-components except of java.awt.Scrollbar, java.awt.Choice and java.awt.List. For these components the Listener Adapter listens to AdjustmentEvent, ItemEvent and ActionEvent. If an internal event occurred, The Listeners Adapter catches and converts it to an external event which is then forwarded to the Collaboration Manager. The Collaboration Manager sends these events to the collaboration server that multicasts them.

Event Adapter

The Event Adapter converts incoming external events to AWT events, which can be processed locally. External events are extensions of the AWT events denoted as remote (external) to let the Listener Adapter be able to differentiate internal occurred events from external (incoming) ones.

3.3 Data Flow

Figure 6 shows the overall event circulation of the system. Applications are embedded in the client as shown in the Figure 6. There are two main data paths in the whole system. The first path is labelled with the number 1,2 and 3. This path is used to send the internal AWT events to the server, which multicasts them to all other participants. The data flow works as follows:

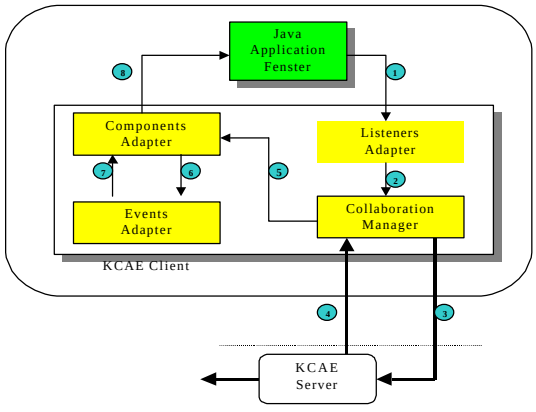


Fig6: Events circulation

1) Any Event occurred in a Java-application is caught by the Listener Adapter.

2) The Listener Adapter tests, whether the event is an external or an internal event. It sends only internal AWT events, which were not received from other clients to the Collaboration Manager.

3) The Collaboration Manager sends incoming events to the server via a RMI connection.

Via the second data path shown in Figure 6 labelled with the numbers 4, 5, 6, 7 and 8, the external AWT events sent from the server are caught by the Collaboration Manager and the Component Adapter in order to reconstruct the event locally. These events are sent to the Java application in the following order:

4) The server sends the remote events to the client.

5) The client catches remote events and sends them to the Component Adapter.

6) The Component Adapter extracts the information about event sources and sends the informations accomplished with the events to the Event Adapter.

7) The Event Adapter converts these events to AWT events and sends them to the Component Adapter.

8) The Component Adapter sends the event to the application which reacts as if the user would interact with the GUI.

4. Collaborative Use of Applets in KCAE

Figure 7 shows the graphical user interface of the KCAE-client applet with a collaboration unaware applet (ItBean Ethernet Frame) used collaboratively.

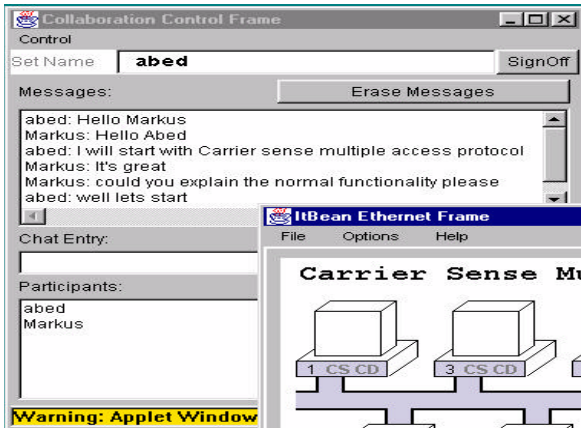


Fig7: The graphical user interface of the KCAE-Client with a collaboration-unaware applet used collaboratively

In general we can summarize the functionality of the

Figure 7) as follows:

- users go to a specified URL (the web server containing the HTML documents and applets to be shared).
- they load an applet (*KCAE-Client*) which offers the following properties:
  - applets list: list of all available applets to be shared. Choosing an applet will cause it to be downloaded to run on the local machine.
  - participants list: contains the names of participants of a running session.
- each interaction with a collaborative applet is transmitted to all participant via the RMI-server.
- users can chat with other participants.
- users may invoke more than one collaborative applet at a time.

5. Related Work

developed a Java Remote Control Tool, which allows to control and synchronize distributed Java applications and applets. The drawback of this approach is that it is necessary to have access to the original source code of the application or applets in order to make it collaborative. That means every applet must initiate a Remote-Control-Client object which is usually done in the constructor of the applet. Also the event handling within the applet must be modified in order to receive and/ or send events from / to remote applets.

The Multimedia Communications Research Laboratory at the University of Ottawa has developed Java Enabled Telecollaboration System (JETS) that supports the development of collaborative applications. JETS [4], [5] is an API, which implies that an application has to be rewritten if it shall run in a collaborative way.

Habanero [6] is an approach that supports the development of collaborative environments. Habanero is in its terms a framework that helps developers to create shared applications, either by developing a new one from scratch or by altering an existing single-user application which has to be modified to integrate the new collaborative functionality. Instead of using applets, which can be embedded in almost every browser, the Habanero system uses so-called

downloaded and installed on the client site.

Java Collaborative Environment (JCE) has been developed at the National Institute of Standards and Technology (NIST) coming up with an extended version of the Java-AWT [8] called Collaborative AWT (C-AWT). In this approach AWT-components must be replaced by the corresponding C-AWT components [11].

All these approaches propose the use of an API, which has the cost of modifying the source-code of an application, re-implementing it or to design and implement a new application from scratch in order to make it collaborative.

Java Applets Made Multiuser (JAMM) [9] is a system which is similar to our approach in terms of its objective: The collaboration of single-user applications. The difference between both approaches is the way how the collaboration is achieved. In JAMM [10] the set of applications that can be shared is constrained to those that are developed using Swing user interface components as part of Java Foundation Classes which are part of the standard JDK since version 1.2.

restricted to those which implement the Java serializable interface.

6. Conclusion and Future Work

In this paper we described a mechanism that enable us to use almost all single-user applets and applications in a collaborative way. We used the replicated approach and keep track of the components to be able to reconstruct user events on the remote side. We hence developed an architecture that allows to collaborate via collaborative-unaware applications without modifying the source code.

We have successfully tested our system on a number of applets, including educational applets (Figure 7) implemented with JDK 1.1, and Swing. A chat functionality is supported directly in our environment. In general, these applets work well with a few minor difficulties related to some limitations described below. A part of these restrictions we encountered is that Frames, Dialogs and FileDialogs created within a collaborative-made application at run time can not be used collaboratively, as long as they are not registered by the Component Adapter explicitly. This leads to the fact that collaboration is only possible for first level windows. Some of the interesting aspect in collaborative environment which we did not consider yet are session control and floor control. Floor control means if a user wants to control the applet, S / he should be able to apply for the control. The collaboration manager on the server sends the request to the owner of the session (if available) to get a permission for handing the control over to the requested user. Losing and gaining a control can be seen as a simple task where gaining the control means

the ability to interact with the whole applet, and losing the control is losing the ability to interact with it. As our architecture allows us to know exactly the components in the applets such a task can be achieved by enabling, disabling the components of the applet if the floor control is gained or lost respectively.

References

[1] J. *JavaBeans Programming from the* Osborne, 1998.

[2] Javasoft Websites: *http://www.java.com/products/rmi*, 1999.

[3] C. Fuhrmann, and G. *Teachware - The Java Remote Control Tool and its* In procceding of ED-MEDIA'98, 1998.

[4] S. Shirmohammadi, J. C. Oliveira, and N. D. Georganas, *"Java-Based Multimedia Collaboration: Approaches and Issues"*, Proc. International Conference On Telecommunications (ICT '98), Vol. I, Porto Carras, Greece, 1998.

[5] S. Shirmohammadi and N. D. Georganas, *"JETS: a Java-Enabled Telecollaboration System"*, Proc. IEEE Conference on Multimedia Computing and Systems (ICMCS '97), 1997.

[6] NCSA Habanero Home Page: *http://www.ncsa.uiuc.edu/SDG/Software/Habanero/* NCSA Software Development Division, 1996.

[7] Java Shared Data Toolkit, *http://www.sun.com/software/jsdt/index.html*, 1999.

[8] H. Abdel-Wahab, J. Favereau, O. Kim and P. Kabore IEEE Computer Society Workshop on Future 1997.

[9] J. Begole, C. Struble, C. Shaffer and R. Smith: . Proceeding of the 1997 Symposium on User NY, 1997.

[10] J. Begole and C. Shaffer,*"Flexible Collaboration Transparency"*, Virginia Tech, Department of Computer Science, Technical Report TR-98-11, 1998.

[11] H. Abdel-Wahab, B. Kvande, S. Nanjangud, O. Kim, J. Favreau, PROMS'96, Madrid, Spain, 1996.

[12] R. Steinmetz, and K. Nahrstedt: , Prentice Hall 1995.

[13] G. Hamilton : , Sun Microsystems, 1997.

[14] J.C. Lauwers, T.A. Joseph, K.A. Lantz and A.L. Romanow, Proceedings of Office Information Systems 1990.

[15] J.Grudin, *Cooperative Work:* , IEEE Computer, Vol.27, No. 5, 1994.