

# Conference Toolkit: A Framework for Real-Time Conferencing

*Amedeo Bonfiglio*

ARG spa

Via Pio la Torre 14, 20090 Vimodrone (MI), Italy

e-mail: argborll@imiclvx.bitnet

*Giuseppe Malatesta*

Olivetti spa

Direzione Olivetti Ricerca, System Software Laboratory

Via C. Colombo 49, 20090 Trezzano s/n (MI), Italy

*Francesco Tisato*

Dipartimento Elettrico, Università della Calabria

Arcavacata di Rende (Cosenza), Italy

## ABSTRACT

This paper introduces Conference Toolkit, a system layer supporting multimedia, real-time cooperation among users via shared applications. Conference Toolkit allows both to integrate standard applications in a conference environment and to develop "conference aware" applications. It is based on a concurrent object-oriented scheme. Conference Desk, a prototype based on the Conference Toolkit model, is described.

## 1. Introduction

In a Computer Supported Co-operative Work system people interact with one another by sharing information. According to Garcia et al. taxonomy [1] the kinds of interaction can be classified as:

- Sharing electronic messages.
- Sharing stored information.
- Sharing application processes.

Interactions based on both messages and data sharing are not real-time, if by "in real time" we mean "within a delay that is negligible in terms of human response time". They are based on a store-and-forward approach. The effects of a user action, e.g. sending a mail or updating a file, are not recognized by another user unless he/she explicitly looks for changes in some information repository. A typical architecture for these kinds of interactions is based on several replicated application processes, one for each user, as shown in Fig. 1. An application process behaves synchronously with its own user and interacts indirectly with other users via a

message system or a shared database. In the following application processes built according to this architectural model will be referred to as *single-user applications*.

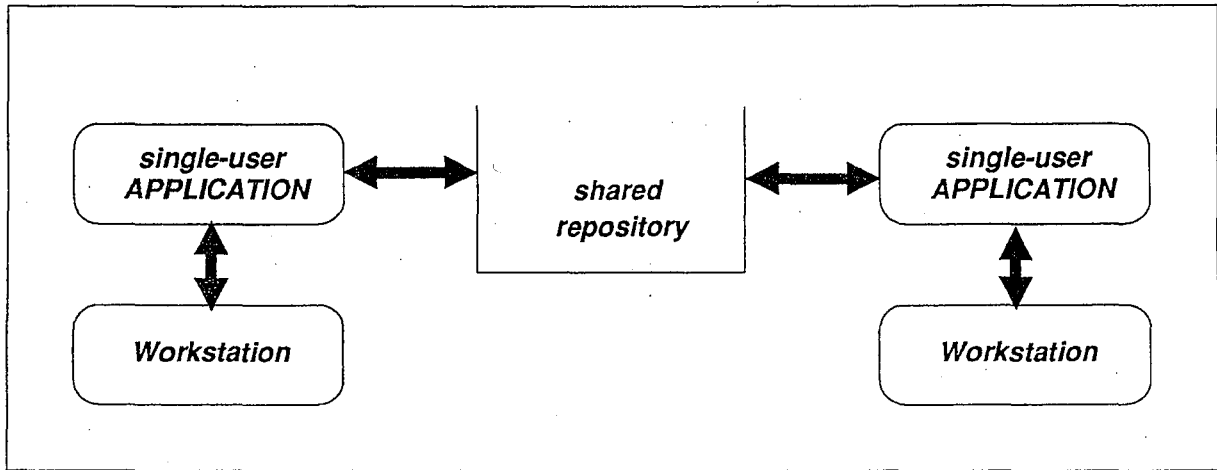


Fig. 1: Single-user Applications

Though these interaction schemes are widely used, they are not satisfactory whenever actions performed by a user must be perceived asynchronously and in real time by its partners. In these cases an architecture based on shared application processes is a valuable solution. A unique process (Fig. 2) interacts with a set of users, so that after a user's input the consequent outputs (possibly including echoes) can be immediately delivered to each user. In the following application processes built according to this architectural model will be referred to as *multi-user applications*. The ultimate case of this scheme is that of a voice phone call, where (after the set-up) the application processing is empty (apart from coding and compression activities, if any).

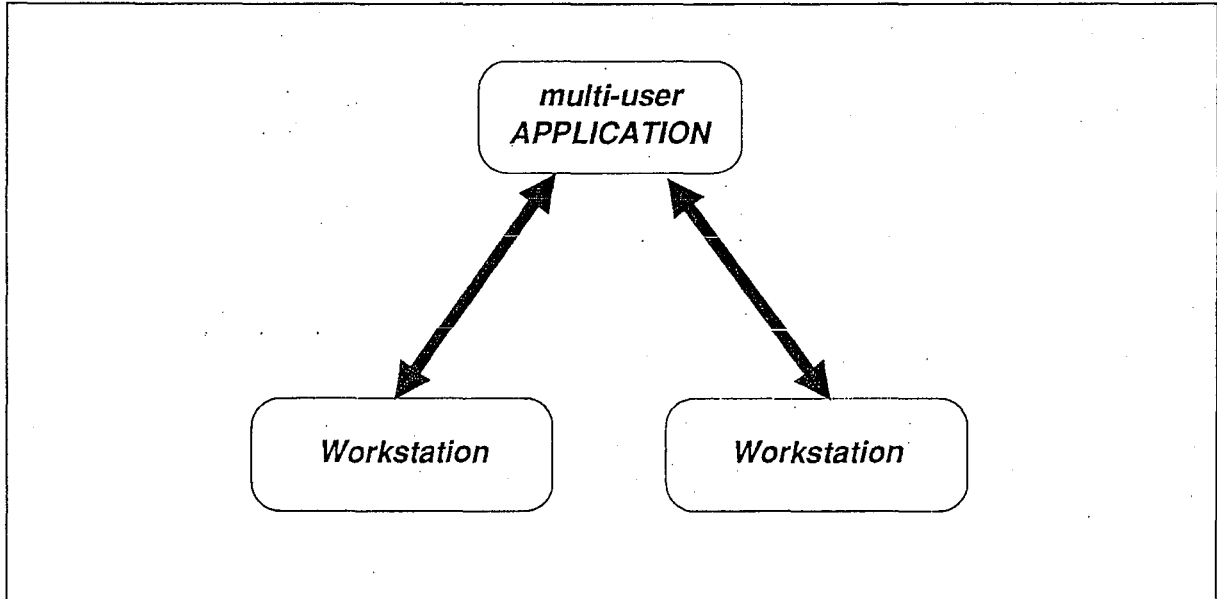


Fig. 2: Multi-user Application

There are several difficulties in the implementation of a multi-user application. First, it relies on the efficiency of the communication subsystem - whose features fall out of the scope of this paper. Second, the application must be able to connect itself to a set of users whose names, locations and capabilities are not a priori known. Third, it must react to asynchronous stimuli generated by the users in an unpredictable way.

Fourth, it must manage floor control strategies to ensure that users interact properly. Fifth, presentation requirements may differ from a user to another one either for logical or for technological reasons. Last but not least, it is mandatory to reuse large and expensive packages (editors, knowledge-base servers, voice servers...) that have been designed and implemented within a single-user philosophy.

Conference Toolkit defines an architectural layer which encapsulates the mechanisms for real-time conferencing. It provides a framework for the development of multi-user applications and for the inclusion of single-user applications in a conference system that can be fully integrated into existing software environments.

Key issues of Conference Toolkit are:

- Distribution* cooperation of users over a network.
- Open System* Pre-existing single-user applications can be invoked within the conference environment without any change. A framework is provided for the development of custom "conference aware" applications.
- Multimedia* Multimedia applications are supported.
- Fine-grain* A typical conference is made of several shared applications. Floor control is treated separately for each application. It is quite different from "electronic meeting rooms" [2].
- Centralization* Only one instance of each shared application is executed and its output is multiplexed to various remote site. This approach is opposite to "Replicated Architecture" [3].
- Policy free* Conference Toolkit does not overimpose any behavioural model so that it can be tailored towards user-specific activities such as brainstorming, group idea processing or cooperative editing.
- Log/play* A conference can be recorded (Log) for being later replayed (Play). Each user can log a conference according to his own view, that is what appears on his display.
- Software platform* Availability of basic networking facilities (e.g. TCP/IP). Operating system/programming environment supporting the construction of distributed software systems as collections of independent modules (processes).
- Hardware platform* High performance workstations interconnected by both local area (e.g. Ethernet) and long-haul networks.

Sec. 2 discusses the process sharing approach by distinguishing between single-user and multi-user applications and by introducing a uniform layered architecture. Sec. 3 introduces the logical model of Conference Toolkit. Sec. 4 presents Conference Desk [4], a working prototype based on the Conference Toolkit approach. Sec. 5 overviews future developments, and mentions related on-going work such as CoAUTHOR [5]. A glance at the Conference Desk appearance is given in Sec. 6.

## 2. Sharing Application Processes

### 2.1. Overview

There are two different ways of building shared application processes:

i) *Extending a single-user application's dialogue to support multi-user interaction.*

This is a key issue, since the reusability of existing application allows to save costs of software developments and ensures consistency with industry standards. Existing *stand-alone* applications (e.g. text and graphics editors) must be plugged into a conference system without any modification. The conference environment is totally transparent to application programs that are executed according to standard practice. Note that since the application is unaware of being shared, it cannot control the flow of information among users. This task is to be performed by a separate system component.

ii) *Writing native "conference aware" multi-user applications.*

This class of applications perform CSCW-specific tasks (i.e. group idea processing or collaborative document production [6]) whose semantics is intrinsically related to the concept of cooperation among users. This cannot be achieved by simply sharing single-user applications and implies to develop "conference aware" multi-user applications.

Though multi-user applications could be designed as monolithic blocks following the scheme of Fig. 2, it is useful to keep as separate as possible the mechanisms which are related to cooperation among users. In the following we show how both single- and multi-user applications can be structured to achieve this goal.

### 2.2. Sharing a single-user Application

Our model postulates the current orientation towards multi-process structuring of user interface software [7] [8]. An application connects via a Data Channel to a possibly remote server process in order to access resources that are local to the workstation (e.g. a window server or, in general, a workstation agent). For instance an application may be connected to the X server [9] and communicate according to the X protocol.

Every application can be conceptually splitted into two major components:

- *Semantical component*, which holds the status of the application, accesses whatever semantical resource it needs (e.g. the knowledge base) and performs conceptual tasks;
- *Presentation, or Dialogue, component*, which mainly implements user interface and communication tasks; it can be considered a sort of "front end" to the Semantical Component.

This logical layering does seldom correspond to a concrete architecture in terms of processes. Fig. 3 exemplifies the case of a typical X-based monolithic multi-user application. The application process is directly connected to the X server and interacts with it via the X protocol. This implies that:

- the application process includes software modules which perform presentation level activities, and depend on the features of the specific server;
- heavy network traffic is produced because the X protocol is low-level and most presentation related events (mouse dragging, expose, resize...) flow through the Data Channel between window server and application process.

Indeed, great advantages can be achieved if the application is actually splitted into its two components,

connected by means of Data Channels and communicating via an application-specific Data Protocol, as shown in Fig. 4.

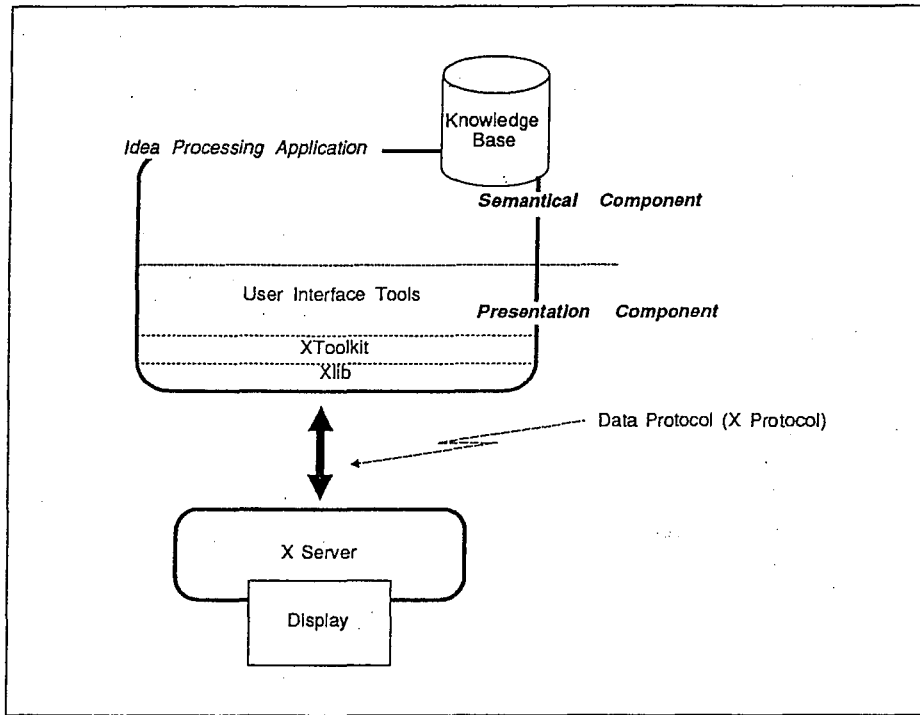


Fig. 3: Monolithic X-application

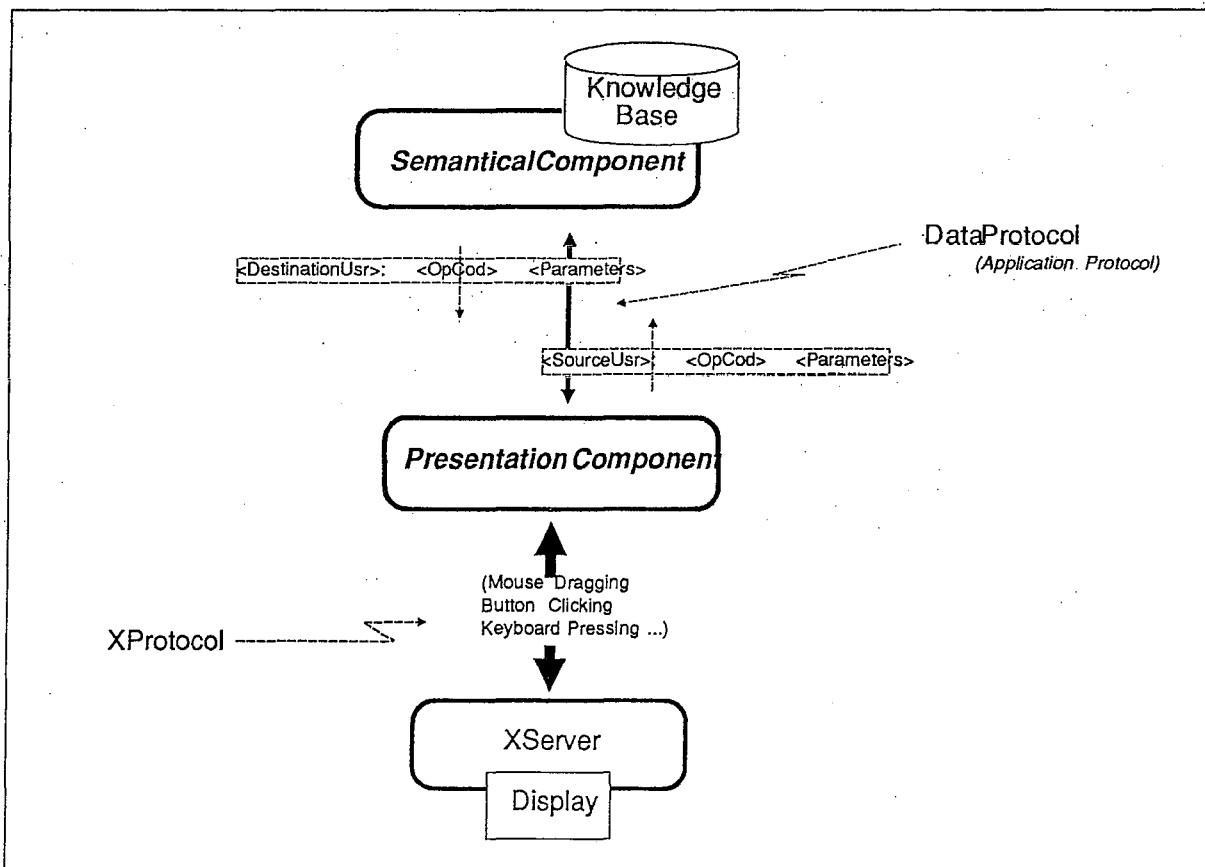


Fig. 4: Splitted X-application

Major benefits of this splitting are the following:

- Network traffic can be greatly reduced since Data Protocol reports only semantically meaningful events.
- Presentation Components provide a *window system independent interface* to their Conceptual Component.
- It is not difficult to achieve *personalized views* of information [10]

Input/output messages of Data Protocol, as sketched in Fig. 4, contain the identifier of the source/destination user. This information could seem redundant for single-user application, but its relevance will become apparent hereafter.

### 2.3. Sharing a multi-user Application

The two-layer model above can be extended in a straightforward way to multi-user applications by introducing an intermediate layer, denoted as *Conference Component* in Fig. 5. It is in charge of:

- setting up the conference, i.e. starting application processes (typically more than one) of the semantic component and monitoring the inclusion of users;
- performing floor control for each application process involved in the conference;
- multiplexing/demultiplexing data streams.

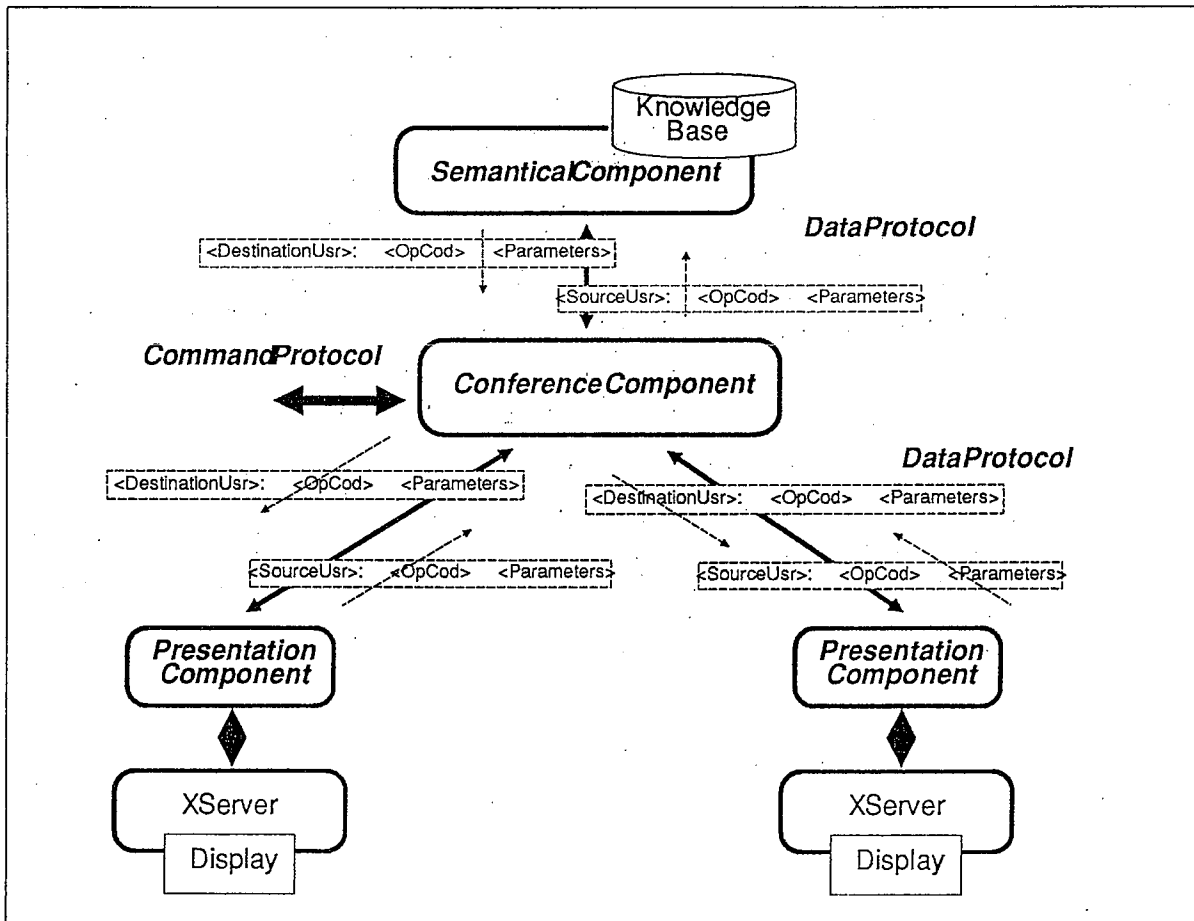


Fig. 5: Conference Component

Multi-user applications discriminate between data streams from/to different users. In the monolithic scheme of Fig. 2 there is, at least conceptually, one Data Channel per user so that input messages can be distinguished on the basis of their source channels. This implies that the application program must deal with system dependent naming schemes. Note that, for multimedia applications, different data streams (e.g. voice and data) could rely on different addressing mechanisms (e.g. sockets and telephone numbers). In our model the Data Protocol, as visible at the semantic level, identifies messages in terms of logical user names. The binding between logical and physical names is encapsulated into the Conference Component.

The Control Protocol defines the set of legal conference control commands and provides a media-independent interface to the Conference Component. As shown in Fig. 5, it is not *a priori* defined which components interact with the Conference Component via the Control Protocol. There are two extreme situations:

i) *sharing a single-user application*. The semantic component is not conference-aware, and control commands are generated by the users interacting with an extension of the presentation component (*Command Filter*);

ii) *fully application-mastered sharing*. Control strategies are totally controlled by the semantic component. Users are in a slave situation.

The former solution is mandatory to reuse standard, single-user packages. In most practical situations suitable intermediate strategies can be built on the top of the basic building blocks provided by Conference Toolkit.

### 3. The Logical Model of Conference Toolkit

#### 3.1. Overview

Conference Toolkit provides a collection of building blocks for the development of Conference Components. The basic building blocks are policy free and can be specialized to fulfill specific requirements of different CSCW environments. Some library building blocks support widespread strategies.

A real-time Conference Component must support:

- Concurrency and asynchronous events management.
- Mechanisms for multiplexing/demultiplexing multimedia information streams.
- Strategies for floor control and for overall conference management.

It is implemented as a collection of concurrent objects communicating via asynchronous messages in a lightweighted-processes framework. Bidirectional logical channels allow to exchange messages both internal and external to the Conference Component.

Conference Toolkit allows to:

- define object classes and instances;
- define logical channels;
- define methods to be invoked whenever an object receives a message.

It provides mechanisms for deriving new classes from existing ones. Specialized Conference Components can be built by deriving new object classes and by defining new methods. Top level, predefined classes corresponding to basic concepts are:

i) *Multiplexer*, providing mechanisms for multiplexing/demultiplexing an information stream.

ii) *Bridge Manager*, controlling a set of logically related multiplexers. By "bridge" we denote a Bridge Manager plus the set of controlled multiplexers.

iii) *Conference Manager*, supporting the overall setup and management of the conference.

Figure 6 sketches the logical view of the Conference System.

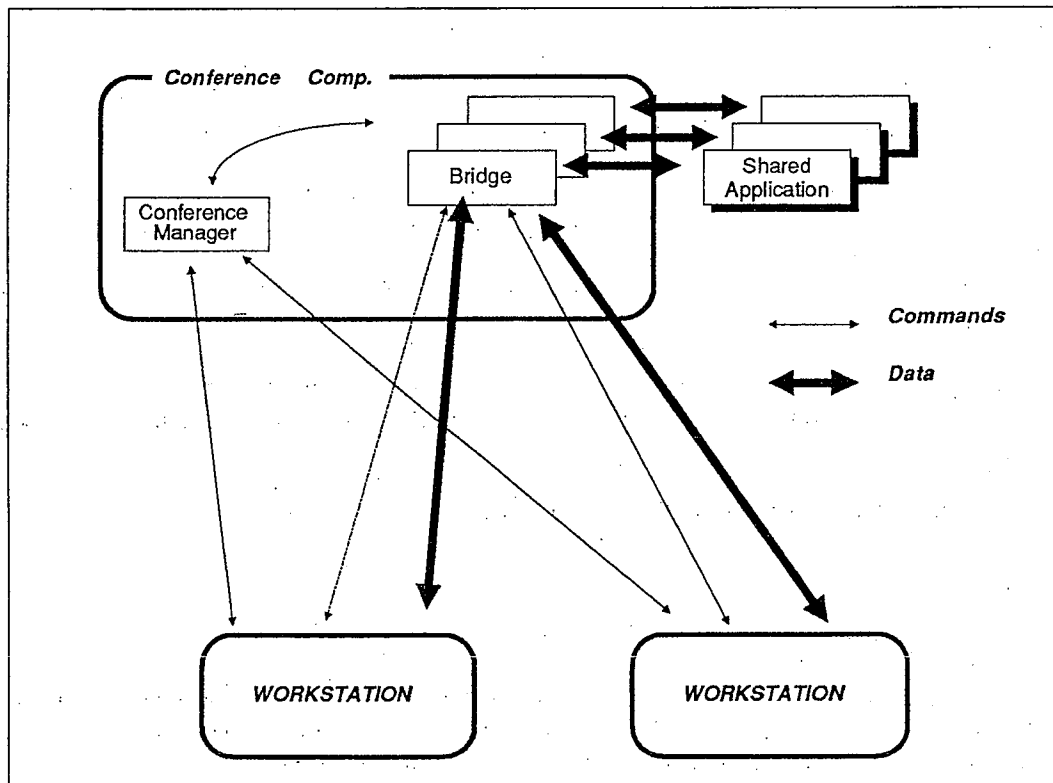


Fig. 6: Logical Model

### 3.2. The Multiplexer

A *Multiplexer* (Mux) multiplexes and demultiplexes data streams between data channels. It is typically devoted to the management of a single-medium data stream. This allows to implement multiplexers that are specialized for the management of a specific medium (e.g. voice) in order to exploit the features of the communication subsystem, which could consist of several disjoint communication media (e.g. voice and data lines). A multiplexer is defined by (Fig. 9):

- The set of currently connected users (*APP\_USR\_set*).
- The set of *Data Channels*. There is typically one Data Channel for each connected user plus one for the application process.
- The *Route Table*, which holds the description of:
  - Data redirection (a sort of "switching matrix").
  - Methods to be activated when receiving data (i.e. encoding, decoding and dispatching).
- The *Control Channel*, typically used by the Bridge Manager to control the Route Table.

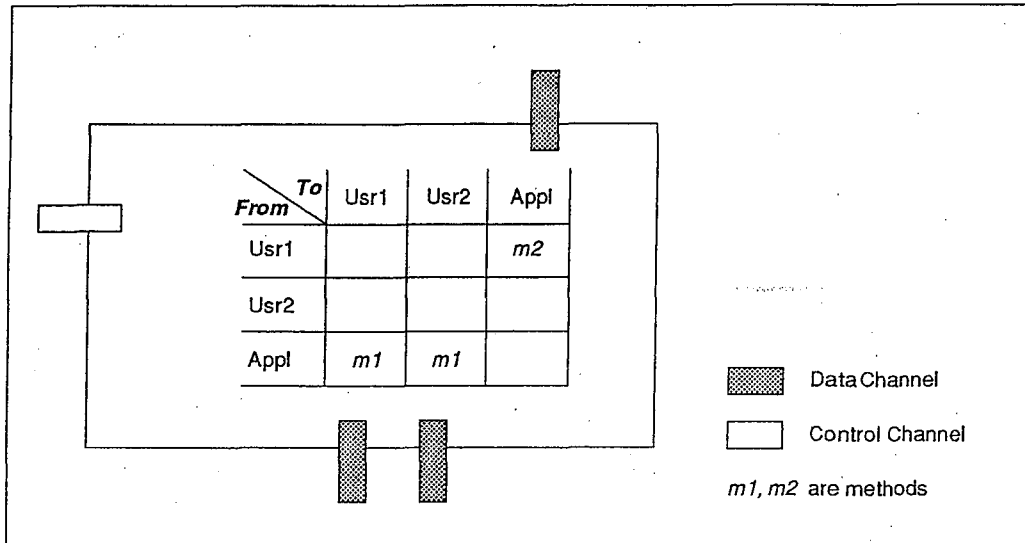


Fig. 7: Multiplexer

Route Table and the associated methods define the behaviour of Multiplexers. They wide from application-independent (mere data dispatchers) to application-specific ones, which interpretate Data Protocol and are aware of application's semantics. Of course, structuring criteria suggest to keep the "semantic content" of a Multiplexer as low as possible.

### 3.3. The Bridge Manager

A conference may imply the execution of several application which evolve autonomously. Each application is in general multimedia, has its own set of users and its own control policy (e.g. blackboard, multiple projector, note pad: see below). A *Conference Bridge* implements the strategies related to a specific application by exchanging commands with the environment according to the control protocol, and by controlling the multiplexing of data.

In terms of objects, a Conference Bridge consists of a *Bridge Manager* and a set of Multiplexers.

The Bridge Manager is defined by:

- The set of currently connected users (*APP\_USR\_set*). A privileged user (the *Application Master*) has special rights over the bridge.
- The set of controlled *Muxes*.
- The *control strategy*, by which the Muxes' Route Tables are set up. Figure 8 reports three different configurations, namely:
  - *Blackboard*, where only one user is allowed to input data, outputs are delivered to all the users. This strategy can be used with a single-user application (e.g. an editor).
  - *Multiple Projector*, where all the users can perform input. This strategy requires a multi-user application.
  - *Mirror*, where there is no application and data from each user are just reflected to all the others. This strategy is typical of voice transmission.
- The set of *current speakers* (*APP\_SPEAKER\_set*), i.e. the users that are allowed to input data according to the floor control strategy.

- The set of *Command Channels* through which bridge commands flow (“Request floor control”, “Request application status”, “Quit application”, etc.). There is usually one command channel for each connected user plus - possibly - one for the application, if it is multi-user.
- Its *Control Channel*, for communication with other objects (in particular, the Conference Manager).

<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <th style="text-align: left;">From \ To</th> <th>Usr1</th> <th>Usr2</th> <th>Appl</th> </tr> <tr> <th>Usr1</th> <td></td> <td></td> <td>m2</td> </tr> <tr> <th>Usr2</th> <td></td> <td></td> <td></td> </tr> <tr> <th>Appl</th> <td>m1</td> <td>m1</td> <td></td> </tr> </table> <p>BlackBoard</p>	From \ To	Usr1	Usr2	Appl	Usr1			m2	Usr2				Appl	m1	m1		<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <th style="text-align: left;">From \ To</th> <th>Usr1</th> <th>Usr2</th> <th>Appl</th> </tr> <tr> <th>Usr1</th> <td></td> <td></td> <td>m2</td> </tr> <tr> <th>Usr2</th> <td></td> <td></td> <td>m2</td> </tr> <tr> <th>Appl</th> <td>m1</td> <td>m1</td> <td></td> </tr> </table> <p>Multiple Projector</p>	From \ To	Usr1	Usr2	Appl	Usr1			m2	Usr2			m2	Appl	m1	m1		<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <th style="text-align: left;">From \ To</th> <th>Usr1</th> <th>Usr2</th> </tr> <tr> <th>Usr1</th> <td>my</td> <td>my</td> </tr> <tr> <th>Usr2</th> <td>my</td> <td>my</td> </tr> </table> <p>Mirror</p>	From \ To	Usr1	Usr2	Usr1	my	my	Usr2	my	my
From \ To	Usr1	Usr2	Appl																																								
Usr1			m2																																								
Usr2																																											
Appl	m1	m1																																									
From \ To	Usr1	Usr2	Appl																																								
Usr1			m2																																								
Usr2			m2																																								
Appl	m1	m1																																									
From \ To	Usr1	Usr2																																									
Usr1	my	my																																									
Usr2	my	my																																									

Fig. 8: Route Tables

### 3.4. The Conference Manager

The Conference Manager controls the overall layout of the conference. It is defined by:

- The set of currently connected users (*CONF\_USR\_set*).
- The set of currently shared applications (*APP\_set*), that is the set of bridges.
- The set of conference attributes such as Log&Play.
- The set of *Command Channels* which are connected to users, one per user, through which conference commands flow (“Start new application”, “Terminate conference”).
- Its Control Channel, used for inter-object communication, typically with bridges (i.e. “User XY has disconnected”).

## 4. A Prototype: Conference Desk

This section surveys Conference Desk, the first prototype of a real-time multimedia conference system based on the Conference Toolkit approach. Appendix 1 will sketch the *look&feel* of Conference Desk.

Conference Desk has been implemented for a network of Sun-3 workstations connected via Ethernet (Sun-3 is a registered trademark of Sun microsystems, Inc.; Ethernet is a registered trademark of Xerox Corporation). A simple voice peripheral is attached to each workstation. It runs under Unix release 3.2 (Unix is a registered trademark of Bell Laboratories) and NeWS release 1.0 (NeWS is a registered trademark of Sun microsystems, Inc.). It is written in C and NeWS-extended PostScript (PostScript is a registered trademark of Adobe Systems, Inc.).

The multi-threaded run-time environment supporting conferencing objects is achieved by means of extensions to C through a preprocessor. NeWS is expected to run on each conferee’s workstation.

For practical reasons the Conference Component of the logical model is splitted into a number of Unix processes, namely:

- The *Conference Engine*. It supports bridges, multiplexers and conference manager. It consists of a logical and a physical layers. The first one handles only logical entities (in particular, channels), while the other one maps the logical world into physical resources (in particular, sockets).
- The set of *Conference Agents*. There is one Conference Agent on each user's machine. It performs a symmetric mapping between logical and physical resources, and controls the local execution of shared applications.

This distributed scheme is network transparent in the sense that Agents must reside on users' machine, while the Engine can be anywhere over the network (see Fig. 9).

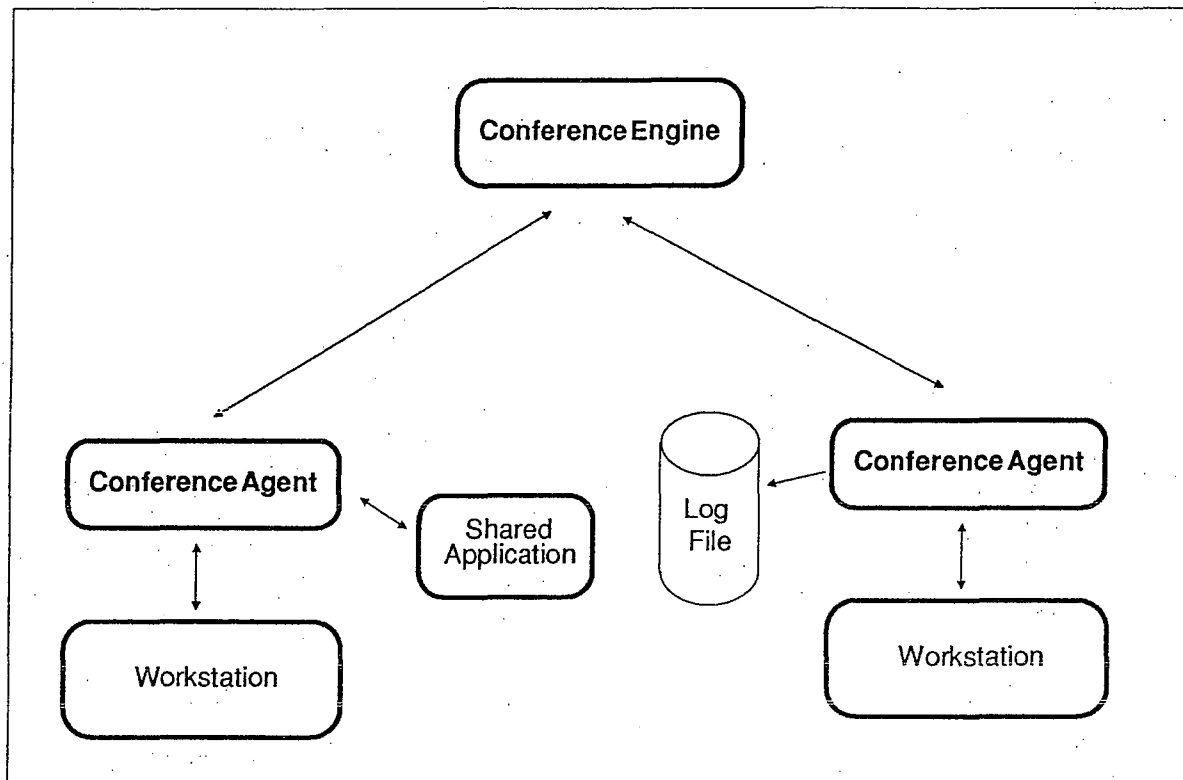


Fig. 9: Concrete Architecture

Each Conference Agent communicates with the Conference Engine on the basis of the Conference Toolkit Protocol which allows to map several logical data and command channels into a unique physical channel. This solution was chosen to solve problems related to the limited number of available sockets.

The implementation of some Data Channels (e.g. voice and video) can exploit the availability of high performance specialized devices. In particular, voice raw data can be transmitted via a token-based network

A shared application is executed on the workstation of the "application-master" user under the control of the local Conference Agent. This allows to run the application in the environment of its application master.

Log&Play is supported by Agents. When Log is active for a user, its Agent record messages from the Conference Toolkit Protocol into a Log file. Play just requests that a Player process mimics the flow of information of the Conference Toolkit Protocol by extracting recorded messages from the Log file and by delivering them to the Agent.

Arbitrary application programs written for NeWS can be plugged into the Conference Desk without modification. This objective proved to be very hard to accomplish because of unpredictable usage of the downloading feature of NeWS protocol.

Two "conference aware" applications, the Voice Blackboard and the Slide Multiple Projector, have been developed as collections of specialized objects on the basis of the building blocks provided by Conference Toolkit. In practice a Voice bridge and a SlideProjector bridge have been derived by subclassing the Blackboard and the MultiProjector class respectively.

## 5. Conclusions and Future Work

We have defined a model for multi-user application sharing within a centralized architecture. We have also implemented a prototype of Conference Toolkit using an environment of Sun workstations connected by Ethernet with a software base of Unix and NeWS. We have demonstrated that in this environment the Conference Toolkit approach is a sound basis for the development of conferencing systems.

We plan to perform some extensions and experiments with Conference Toolkit.

- To add support for applications that use the X protocol. At the date, a new class of multiplexer is being developed, the *Xplexer*, in order to support the sharing of single-user X applications: thus Conference Toolkit could support both X and NeWS applications at the same time, by means of *Xplexer* and *NeWSplexer* respectively, provided that both window systems are available on the same machine [11].
- To extend the set of available classes of multiplexers in order to support new functionalities and new media.
- To implement new user interfaces for Conference Toolkit; currently the user-interface has been developed upon Intrinsic, but other related works are investigating with more sophisticated tools (i.e. Andrew).
- To experiment with "multiple views" of information of the same multi-user application program.
- To develop interactive tools for supporting the definition of new conferencing building blocks.

Conference Toolkit is part of MULTWORKS (MULTimedia Integrated WORKStation), a recently started ESPRIT-II Technology Integration Project (No. 2105). A number of application programs, in the area of Computer Supported Co-operative Work, are currently under development; in particular, CoAUTHOR [5], a hypermedia-oriented real-time group authoring environment. The final goal is full integration with the real-time-conference framework provided by Conference Toolkit.

The authors are greatly indebted to Stefano Piccardi and Carlo Serrelli, whose contribution has been invaluable during the definition of Conference Toolkit and the implementation of Conference Desk.

## 6. Appendix 1. Glancing at Conference Desk

This appendix attempts to show some features of Conference Desk.

The last full-page figure shows Jerry's desktop during a conference with Tom. Tom's desktop is not shown, but it looks very much like Jerry's, apart from a different window arrangement and a few extra-conference windows.

The conference control panel (conference window) is shown in the lower left corner of the desktop. Conference Desk is displaying two additional windows - a conference status window and a dialogue window, for typing input - in the left side of the desktop. Several help windows can be accessed through the button labelled "Info on Conference Desk".

Conference Status Window: demo

NOW ENABLED TO INPUT jerry  
 CONNECTED TO THIS BRIDGE.  
 jerry (Master) (Input)  
 ECM  
 REQUESTING THE FLOOR  
 ECM

v100 terminal emulator

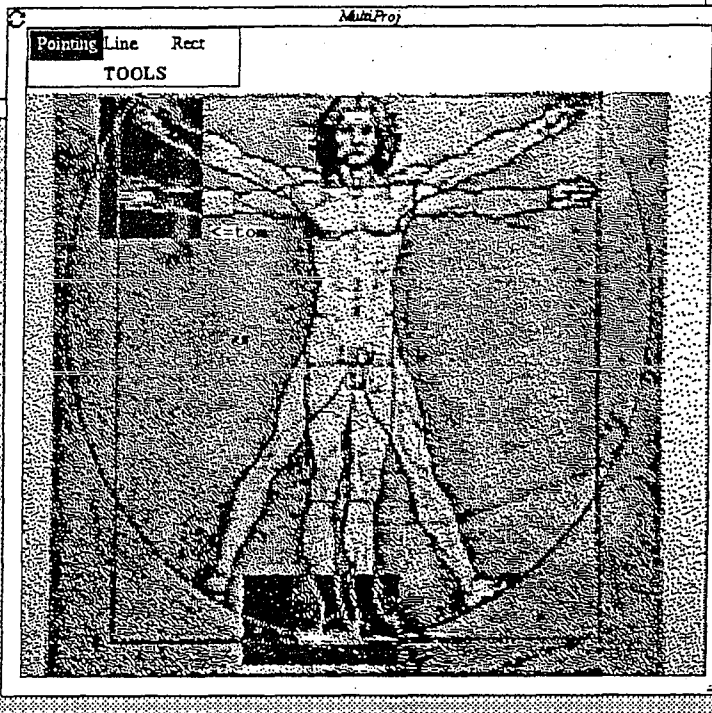
Tom, would you please highlight the  
 damaged spots in Leonardo's Man...

Conference =>	Master is Jerry
Move	Request Floor Control (F2)
Move Constrained	Relinquish Floor Control (F3)
Top	Relinquish & Request (F4)
Bottom	List Floor Requests
Zap	
Redze	Resume Floor Control
Stretch Corner	Top Floor Queue
Stretch Edge	
Close	Fix Bridge Display (F1)
Redisplay	Delete Bridge
	Bridge Status

Dialog Window: demo

Partner's Name

Delayed Insertion (on/off)



CONFERENCE WINDOW: demo

MASTER IS: jerry Terminal Emulator accepted: Please wait.

Partner List Stop Log Trash Log

Applications Multiple Projector Voice

Info on Conferences Desk Terminate Conference

Tom and Jerry are sharing two applications: the NeWS terminal emulator `psterm` (right top) and a special conference application called "Multiple Projector" (right bottom). The Multiple Projector is used to line on, to highlight regions of, and to point at the underlying canvas (notice the arrow labeled "tom" under Leonardo's *Man's* right arm).

Jerry has just used the terminal emulator to tell Tom to highlight damaged regions of the drawing. Jerry could also use the "Voice" button to initiate a voice conversation with Tom. Tom has highlighted the regions and is now pointing at one of them. All conferees can freely drag their pointers concurrently.

Recall that applications are shared via *bridges*. In this example, there are two active bridges that provide a multi-user interface to the two applications. By selecting the "Bridge Status" menu entry of the conference submenu in an application, it is possible to know the status of the bridge attached to that application. In this moment, the conference status window is showing the status of the bridge attached to the terminal emulator; Tom and Jerry are "connected to this bridge", i.e. both receive terminal emulator output; Jerry is the initiator of the terminal emulator (Master), and his keyboard input is currently accepted (Input), i.e. he has the floor; Tom has himself requested the floor.

## 7. References

- [1] J. J. Garcia-Luna-Aveces, Earl J. Craighill, Ruth Lang, "An Open-system Model for Computer-Supported Collaboration", In *Proc. 2nd International Conference on Computer Workstations*, IEEE, March 1988, pp.40-51.
- [2] J. Robert Ensor, S. R. Ahuja, David N.Horn, S. E. Lucco, "The Rapport Multimedia Conferencing System- A Software Overview", In *Proc. 2nd International Conference on Computer Workstations*, IEEE, March 1988, pp.52-58.
- [3] Keith A. Lantz, "An Experiment in Integrated Multimedia Conferencing", In *Proc. CSCW '86: Conference on Computer-Supported Cooperative Work*, MCC Software Technology Program, December 1986, pp.267-275.
- [4] S. Piccardi, F. Tisato, "Conference Desk - an Experiment and a Model for Application Sharing", Jan 1989, Int. Report, Olivetti D.O.R. Milano.
- [5] U. Hahn, M. Jarke, K.Kreeplin, M Farusi, F. Pimpinelli, "CoAUTHOR, A Cooperative Group Authoring Environment", 1989.
- [6] Mary D. P. Leland, Robert S. Fish, Robert E. Kraut, "Collaborative Document Production Using Quilt", ACM, 0-89791-282-9/88/0206, pp. 206-215.
- [7] Keith A. Lantz, Peter P. Tanner, Carl Binding, Kuan-Tase Huang, Andrew Dwelly, "Reference Models, Window Systems and Concurrency", *Computer Graphics*, 21(2), April 1987, pp. 87-97.
- [8] Keith A. Lantz, "Multi-process Structuring of User Interface Software", *Computer Graphics*, 21(2), April 1987, pp. 124-130. Presented at the SIGGRAPH Workshop on Software Tools for User Interface Development, November 1986.
- [9] Robert W. Scheifler, Jim Gettys, "The X Window System", *ACM Transactions on Graphics*, 5(2), April 1986, pp. 79-109.
- [10] Stefik M., Bobrow D.B., Foster G., Lanning S., Tatar D., "WYSIWIS revised: Early Experiences with Multi-user Interfaces", *ACM Transaction on Office Information Systems*, 5, 1987, pp. 147-167.
- [11] Robin Schaufler, "X11/NeWS Design Overview", Sun Microsystems, Inc.