4

Contract N00014-86-K-0680

# A Prototype for Remotely Shared Textual Workspaces

*Hussein M. Abdel-Wahab*
Department of Computer Science
North Carolina State University
Raleigh, NC 27695

*Sheng-Uei Guan and J. Nievergelt*
Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, NC 27514

## SUMMARY

Computer-based collaboration between geographically dispersed users is still limited primarily to electronic mail and file transfer, but there is increasing interest in computer support for real-time interaction between remote users. The problem of implementing remotely shared workspaces, which allow users to operate simultaneously on the same objects, is of broad interest. Our objective is to show textual workspaces that allow real-time collaboration can be implemented efficiently by using existing operating systems and communications primitives. This paper documents our experiences in implementing a prototype under Berkeley UNIX[1], using the programming language C, and Berkeley Interprocess Communication facilities. We describe design alternatives that take into account the communications bandwidth between the different sites of the network, and we introduce an efficient protocol that regulates user access to the shared workspace. [illegible]

**KEY WORDS:** Distributed systems, Real-time collaboration UNIX
Interprocess communications.

[1]UNIX is a trademark of AT&T Bell Laboratories

## REMOTELY SHARED WORKSPACES: PROBLEM AND APPROACHES

In everyday life individuals meet to review and edit documents that include text and pictures. The main objective of our work is to provide usable low-cost alternatives to physical meetings held for the purpose of editing and reviewing documents. Every participant in this "virtual" meeting has in front of him a workspace where he can operate on the same objects that other participants see in their workspace. Just as a conference telephone call provides a shared "audio" space to the participants, the systems we are describing provide a shared "visual" workspace. Even if the quality of human interaction is not as high in the remote virtual meeting as it is in the physical meeting, as long as it is above some threshold of acceptability, economic considerations may well favor a remote meeting. With the proliferation of high bandwidth computer networks, and the increasing popularity and affordability of powerful workstations, it is feasible to provide the users with the environment and the tools to achieve this goal.

Today, computer-based collaboration between geographically dispersed users is still limited primarily to electronic mail and file transfer, but several research groups experiment with more powerful computer support for team work[1,2,3]. Let us summarize the features of some of the prototypes described in the literature.

At the MIT Laboratory of Computer Science, users share information from their personal calendars to schedule meetings using a real-time conferencing system, RTCAL[4]. Participants speak to each other over the phone and use their workstation display as a blackboard. Another system, CES, is a collaborative editing system for co-authors working asynchronously on a shared structured document, where users can work independently on separate sections of the document[5].

At the Xerox PARC Intelligent Systems Laboratory, an experimental meeting room, Colab, provides computational support for collaboration in face-to-face meetings. Several prototype collaboration systems have been built using Colab, such as WYSIWIS (What you see is what I see) which is a multiuser interface that expresses many of the characteristics of a chalkboard in face to face meetings[6,7].

At IBM, an asynchronous conference system has been implemented for Bitnet and Vnet[8]. It is not based on real-time interaction, users send and receive information at their own convenience. At Stanford, an experimental multimedia conferencing facility has been implemented using the V-system, a message-based distributed operating system[9].

In summary, research on computer support for people working together simultaneously, where real-time interaction is essential, is still in its beginnings. Most work is based on experimental systems that exist only in one laboratory. In contrast to the prevailing trend, our objective is to build usable, low-cost remotely shared workspaces based on widely available systems and single-user applications programs, in order to allow a large user community to experiment with this new technology of real-time collaboration. Our approach differs from other collaborative tools in that we offer a general purpose utility that converts any single-user tool into one that can be used for real-time collaboration among several remote users.

This paper documents our experience in constructing a prototype for remotely shared textual workspaces, with the intent to demonstrate the feasibility of implementing such a system quickly using widely available software and hardware resources. We use the C programming language and the UNIX system calls available under the 4.3 version of Berkeley. The users interact with the system using a software tool such as a text editor, the only learning overhead is that of mastering a few simple commands for session control.

The paper is organized as follows. We start by comparing two alternative system designs; present details of a protocol for regulating user access to the shared workspace; describe the user interface and the prototype implementation. We conclude with open issues and an outline of our future goals.

## DESIGN ALTERNATIVES

Fig. 1 depicts a collaborative session between several users that share and operate simultaneously on "objects" contained in a "workspace". Examples of objects are text files, graphs, or images. A workspace is a container for objects: a data management system for creating, accessing, and displaying objects. The basic mode of operation is *what you see is what I see*: all participants have identical views of the objects. Users manipulate objects using "tools", i.e. single-user application programs that can be used to operate on the objects. For example, a text editor is an appropriate tool for text file objects. The basic entities of our model: users, objects and tools, may reside at arbitrary sites of a computer network. We have implemented a prototype system of remote shared workspaces where objects are restricted to be text files, and tools can only accept input from keyboards. For such "textual workspaces", participants may use different types of terminals or workstations.

If two or more participants can issue commands simultaneously, serious problems may occur. For example, assume a text editor like *vi* is used to edit a text file that contains the line:

```
.... the boy boy ran fast ....
```

If two users simultaneously issue the "dw" command to delete the duplicate word, we end up with:

```
.... the ran fast ....
```

To make sure that at any one time, only one user can issue commands (the "active user"), we introduce the synchronization concept of a *token*: the tool accepts input only from the active user, the one who currently holds the token. The token circulates between the participants according to a fair and efficient protocol described in the next section.

In addition to joint manipulation of objects, participants need to manage a session (creating it, selecting objects, joining and leaving a session, etc.). Thus it is natural to introduce two windows: a "tool window" $W_t$ for input and output from the tool, and a "control window" $W_c$ for session management, including token control. Windows can be created using any available window management system such as the Berkeley UNIX 4.3BSD *window* program that runs on any ASCII terminal, or the MIT *X-windows* that runs on a variety of workstations including *SUNs* and *DEC VAXs*.

Participants should be able to discuss what is being displayed on their screen. An intensive discussion may require a telephone conference call, but occasional comments and questions can be exchanged via the control window $W_c$, which doubles as a "talk window".

For terminals with different physical characteristics, users should be aware that the terminal with the smallest tool window determines what operations can be followed conveniently by all users. All participants define a tool window $W_t$ with the same number of lines and characters per line. No inconvenience results from control windows $W_c$ of different sizes.

Thus far we have discussed the functional design of the system: users see a single shared workspace through the windows on their terminal. Possible implementations of this design differ according to the degree of (de-)centralization and replication of the physical storage of the workspace objects and of the executable code for the tool. Among many alternatives, we have implemented two designs that appear to be most natural.

Fig. 2 shows a centralized implementation of workspace and tool. Every participant is represented by a "user agent" process $P_u$ that handles both his tool window and his control window (merges input from the windows and distributes output to them). The centralized data and processes may reside on a separate computer or on one of the participant machines. Fig. 3 shows a replicated implementation where every participant has his own copy of workspace and tool. This makes sense when each participant runs on a separate computer.

In both figures, processes $P^*$ called the *session server process* is connected to all $P_u$ processes (we choose '*' as a superscript of P, since the process is directly connected to other processes in a "star" like topology). The following is a description of the major functions performed by $P_u$ and $P^*$ in each approach:

In the centralized model of Fig. 2, process $P_u$ accepts input typed by the active user in the tool window $W_t$ and sends it to $P^*$ which forwards it the tool process. Process $P^*$ reads the output generated from the tool process and send it to all $P_u$ processes for display in $W_t$. In this model, we assume a high bandwidth between the session server process $P^*$ and the participant sites in order to carry out the usually large volume of data generated by the tool.

If the bandwidth is not sufficient to provide a reasonable response time, then we may use the replicated model shown in Fig. 3. In this model, process $P_u$

reads the input typed in window $W_t$ by the active user and sends it to process $P^*$, which in turn broadcast it to every $P_u$ connected to it. Upon receiving the input, $P_u$ forwards it to the local tool process. Thus every input typed by the active user is given to every tool process via $P^*$. The voluminous output from the tool is displayed locally at each site. Therefore in the replicated model, only the "keystrokes" need to travel through the network. This may lead to better response for remote users connected to the session with low bandwidth channels. At the begining of the session a copy of the original workspace objects is sent to each participant, and at the end of the session all copies are deleted; only the workspace objects at the "session creator" site is retained. A major challenge of this approach is to keep all workspaces mutually consistent throughout the duration of the session.

Input to the control window $W_c$ is handled in the same way in both models. Process $P_u$ receives the input of any user (not only the active user) and process it according to its type, e.g., if it is a token control command, appropriate messages are send to $P^*$ and to the control window $W_c$. In the next two sections we describe in details the protocols and procedures for dealing with $W_c$ input.

## TOKEN MANAGEMENT PROTOCOL

How can users share access to the tool efficiently and fairly? For effective work, the active user must be guaranteed an uninterrupted quantum of time, $Q$, once he gets the token. For fairness, other participants must be able to request the token and obtain it within a certain known waiting time. When the active user has to release the token, he is given a brief grace period, $G$, to complete his current task. Values for $Q$ and $G$ can be set when a session is created, depending on the tool and the number of users. Current default values are: $Q$ = 30 seconds, $G$ = 5 seconds. Three token management commands:

`GET_TOKEN`, `RELEASE_TOKEN` and `FIND_STATUS`

are entered by typing `CTRL-G`, `CTRL-R`, and `CTRL-F`, respectively, in the control window $W_c$, where system responses are displayed.

`GET_TOKEN`: The system will react to the requesting user ("R-user") as follows:

1. If the token is free, the R-user gets it and given the message :

   ```
   You have the token
   ```

   All other participants receive the message:

   ```
   R-user has the token
   ```

2. If the token is being held by another user (the "active user"), the following message is sent to the R-user:

   ```
   please wait
   ```

   The request is inserted into a FIFO "token queue" maintained by the session server $P^*$. When the token becomes available, it is given to the user at the head of the queue. If the queue is non-empty the message:

`please release the token within` $Q$ `seconds`

is sent to the active user. If the active user does not release the token after $Q$ seconds, he is sent the message

`token will be seized in` $G$ `seconds`

if the token is not released by the end of the grace period, the user is given the message:

`time expired, token seized`

3. If the R-user is waiting in the token queue, he will get the message:

`please be patient`

`RELEASE_TOKEN`: If issued by the active user, makes him inactive. If issued by a user waiting in the token queue, deletes his entry from the queue. If the queue becomes empty after deletion, then the "release token" message is retracted by displaying:

`token request canceled, you may continue`

`FIND_STATUS`: The active user name and the queue status are displayed.

Fig. 4 shows the state diagram for a user process $P_u$ with respect to the token. Transitions from one state to another depend: on the clock, on token management commands entered by the user, and on messages received from the server process $P^*$. Each transition is labeled by a pair *event/action*. An *event* is either a user input in $W_c$, an expired time, or a message received from $P^*$. An *action* is either a message send to $P^*$ or a message displayed in $W_c$.

## USER INTERFACE AND PROTOTYPE IMPLEMENTATION

To use our system, one user (called the "session creator") initiates a session by typing:

```
% create_session
```

This creates the session server process $P^*$ which prompts the user to provide the following information:

- *participant names:* the login names of all participants,

  e.g., `wahab guan jn`

- *object names:* the file names to be included in the workspace.

  e.g., `intro, sec1, append`

- *tool name:* the tool name to be used during the session

  e.g., `vi`

- *model type:* specify whether a "centralized" or "replicated" model will be used.

$P^*$ then displays an integer, the ***session-id***, to be used by all participants in establishing connections with $P^*$. Note that the session-id is needed since there may be more than one concurrent session on the machine running $P^*$, and participants need to identify the session(s) they wish to join. Process $P^*$ waits for all participants to be connected and join the session.

To join a session, a user must be on the participant list, otherwise any attempt to join the session is rejected by $P^*$. Each user creates the two windows $W_t$ and $W_c$, using the Berkeley 4.3BSD *window* program (or any other window management system).

For each window there must be an active process that reads input typed in the window and send it to the user-agent process $P_u$, and displays the messages directed to the window by $P_u$ as shown in Fig 5 (a). In our implementation, in order to save one process, we decided to let process $P_u$ runs in window $W_t$ as shown in Fig. 5(b). In the following we refer to process $P_c$ running in the control window $W_c$ as the "chat" process.

The user creates process $P_u$ in the tool window $W_t$ by typing:

```
% user-agent
```

$P_u$ asks the user to provide the following information:

- *host name:* the name of the machine where $P^*$ is running
- *session-id:* the session_id displayed by $P^*$ when it started.

Process $P_u$ displays an integer called *port-id* and waits for a connection by $P_c$. Switching to the control window $W_c$, the user types the command:

```
% chat port-id
```

where *port-id* is the number displayed by $P_u$ in window $W_t$. This creates process $P_c$ and uses the port-id to establish a connection with $P_u$. Soon after this connection is established, process $P_u$ is connected to $P^*$ using its host name and session-id.

When all participants on the list have been connected, process $P^*$ informs each process $P_u$ about the mode of operation: *central* or *replicated*. In central mode, $P^*$ creates the tool process (see Fig.2), while in the replicated mode, $P^*$ sends a copy of the workspace objects to each participant, and each participant runs his own tool process (see Fig. 3).

When the session starts, the active participant uses window $W_t$ to interact with the tool in the way described in the tool manual, as if the active user were the only user. Tool commands from other users are rejected with a warning message sent to their control window $W_c$.

In addition to the "Token Control Commands" discussed earlier, the following "Session Control Commands" are available for participants to interact with the session process $P^*$:

`ESC-END`: to end the session by closing files and communications channels.

`ESC-STAT`: to display information about the session, such as: who are the participants, the objects in the workspace, when the session has started, etc.

The control window $W_c$ can also be used for conversation between users. Any line typed in a control window that does not start with CTRL or ESC characters will appear on all other $W_c$ windows preceded by the name of the writer.

## Interprocess Communications

Communication between processes, e.g. between $P_u$ and $P^*$, is based on the 4.3 BSD *stream sockets* in the *inet domain*[10]. This provides us with a reliable, bidirectional virtual circuit connection between these unrelated processes running at the same or at different machines of the network. For a complete description of Berkeley UNIX Interprocess Communications facilities, see Reference 10 and for an introduction see Reference 11. In our implementation, process $P^*$ creates a *socket*, *bind* it to a *port* and *listen* for connections to be made by processes $P_u$. In turn, each process $P_u$ creates a socket, bind to port and listen for connection to be made by process $P_c$ in the same machine.

Processes use the *SELECT* system call to multiplex input from several sources[10]. For example, process $P_u$ need to monitor the input from the socket connecting it to process $P_c$, the standard input from window $W_t$, the socket connecting it to process $P^*$ and, in case of replicated model, the output from the tool process. Here we should mention that many tools will not function properly without a terminal for stdin and stdout, and in order to let these tools read from and write to other process rather than a terminal, a "pseudo terminal" device is used between the tool and these processes[10]. For example, we have used a pseudo terminal between $P^*$ and the tool in the central model, and between $P_u$ and the tool in the replicated model.

## Frame Types and Processing

Since the virtual circuit between process $P_u$ and $P^*$ will carry messages from different sources, messages are packaged in *frames* of the format:

| TYPE | LENGTH | DATA |
|---|---|---|

Frames are of the following types:

token: for token control, such as "get token" etc.

session: for session control, such as "end session" etc.

chat: for conversation messages between participants.

tool: for input to or output from the tool.

**Frame processing in $P_u$** : Frames are processed in $P_u$ as follows:

1. if the user has the token, the input typed in window $W_t$ is sent to $P^*$ in a tool type frames.

2. data received from $P_c$ is processed according to the following cases:

   - data starting with CTRL character is interpreted and a token type frames is sent to $P^*$ and $P_u$ will change its state accordingly.
   - data starting with ESC character is checked and sent to $P^*$ in a session type frames.
   - any other data is sent to $P^*$ in a chat type frames.

3. data received from $P^*$ is acted upon as follows:

   - token type frames is interpreted and a message is sent to process $P_c$ for display in window $W_c$. $P_u$ will change its state accordingly.
   - data in chat type frames is sent to $P_c$ for display in window $W_c$.
   - session type frames is interpreted and cause an appropriate actions, e.g, terminate a process.
   - data in tool type frames is processed based on the following two cases:

     For replicated model: forward to the local tool process.

     For central model: display in the tool window $W_t$.

**Frame processing in $P^*$** : Frames are processed in $P^*$ as follows:

1. token type frames received from $P_u$ are interpreted and handled according to the token management protocol described earlier.
2. Received frames of type chat is forwarded to every $P_u$ process, except the one it has come from.
3. tool type frames are processed according to the following cases:
   - For replicated model: Broadcast the received frames to all $P_u$ processes,
   - For single model: Forward the data to the tool process. The output from the tool process is sent to all $P_u$ processes in tool type frames.
4. session type frames are processed based on its meaning, for example, for an end of session message, various housekeeping operations are performed.

## Source Code

The system implemented consists of three main programs:

- session server ($P^*$): 771 lines of C code.
- user agent ($P_u$): 796 lines of C code.
- chat ($P_c$): 185 lines of C code.

In addition, the common declarations and subroutines are 293 lines of C code. Write to the authors for a copy of the source.

## CONCLUSIONS, FUTURE GOALS

A simple remotely shared workspace system has been implemented using the UNIX interprocess communication primitives. There appear to be three major directions for research that promise to make collaborative teamwork a more effective and widely used tool in the future:

1. Human factors studies as to what features and functions are valued and actually used by different users working on different tasks. For example, insisting that all users work on the same, identical, text imposes severe synchronization constraints that may slow down the work unnecessarily. When users are happy with the model that they work on different versions of the same object, versions that need not be identical at all times, less constraining implementations become possible.

2. Studies of concurrency, consistency and synchronization that are particularly suited to supporting real-time interaction among several people. Most of the concepts and techniques known today were developed in response to problems of machine-machine interaction or human-machine interaction. It is conceivable that novel synchronization problems and techniques will arise in the context of human-human interaction via the intermediary of a machine.

3. The implementation of remotely shared graphics workspaces. This raises problems of dealing with a greater variety of interactive I/O devices that tend to differ from machine to machine.

Acknowledgements

We are grateful to Peter Calingaert, John Menges, John Smith, Michael Stumm, and the referees for helpful comments. This work was partly supported by ONR under contract N00014-86-K-0680.

## REFERENCES

1. I. Greif, 'Computer Support for Cooperative Office Activities', *Proceedings of the 1982 Office Automation Conference,* San Francisco (April 1982).

2. S.K. Sarin, 'Interactive On-Line Conference', *Ph.D. Thesis, MIT,* Also available as Laboratory for Computer Science Technical Report MIT/LCS/TR-330 (1984).

3. G. Foster, 'Collaborative Systems and Multi-user interfaces', *Ph.D. dissertation, Division of Computer Science,* University of California, Berkeley (1986).

4. S. Sarin, and I. Greif, 'Computer-Based Real-Time Conferences', *IEEE Computer 18,10* Special issue on Computer-Based Multimedia Communications. 33-45 (October 1985).

5. R. Seliger, 'The Design and Implementation of a Distributed Program for Collaborative Editing', *Master Thesis, MIT,* Also available as Laboratory for Computer Science Technical Report MIT/LCS/TR-350 (1985).

6. M. Stefik, D. G. Borbrow, S. Lanning, D. Tatar, and G. Foster, 'WYSIWIS revised: Early Experience with Multi-user interfaces', *Proceedings of the conference on Computer-Supported Cooperative Work,* Austin Texas 276-290 (December 1986).

7. M. Stefik, G. Foster, D. G. Borbrow, K. Kahn, S. Lanning, and L. Suchman,

'Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings', *Communications of the ACM 30,1* 32-47 (January 1987).

8. N. Jarrell, and W. Barrett, 'Network-based Systems for Asynchronous Group Communication', *Proceedings of the conference on Computer-Supported Cooperative Work,* Austin Texas, 184-191 (December 1986).

9. K. A. Lantz, 'An Experiment in Integrated Multimedia Conferencing', *Proceedings of the conference on Computer-Supported Cooperative Work,* Austin Texas, 267-275 (December 1986).

10. S.J. Leffler, R.S. Fabry, W.N. Joy, P. Lapsley, S. Miller, and C. Torek, 'An Advanced 4.3BSD Interprocess Communication Tutorial', *Computer System Research Group,* Department of Electrical Engineering and Computer Science, University of California, Berkeley (1986).

11. S. Sechrest, 'An Introductory 4.3BSD Interprocess Communication Tutorial', *Computer System Research Group,* Department of Electrical Engineering and Computer Science, University of California, Berkeley (1986).

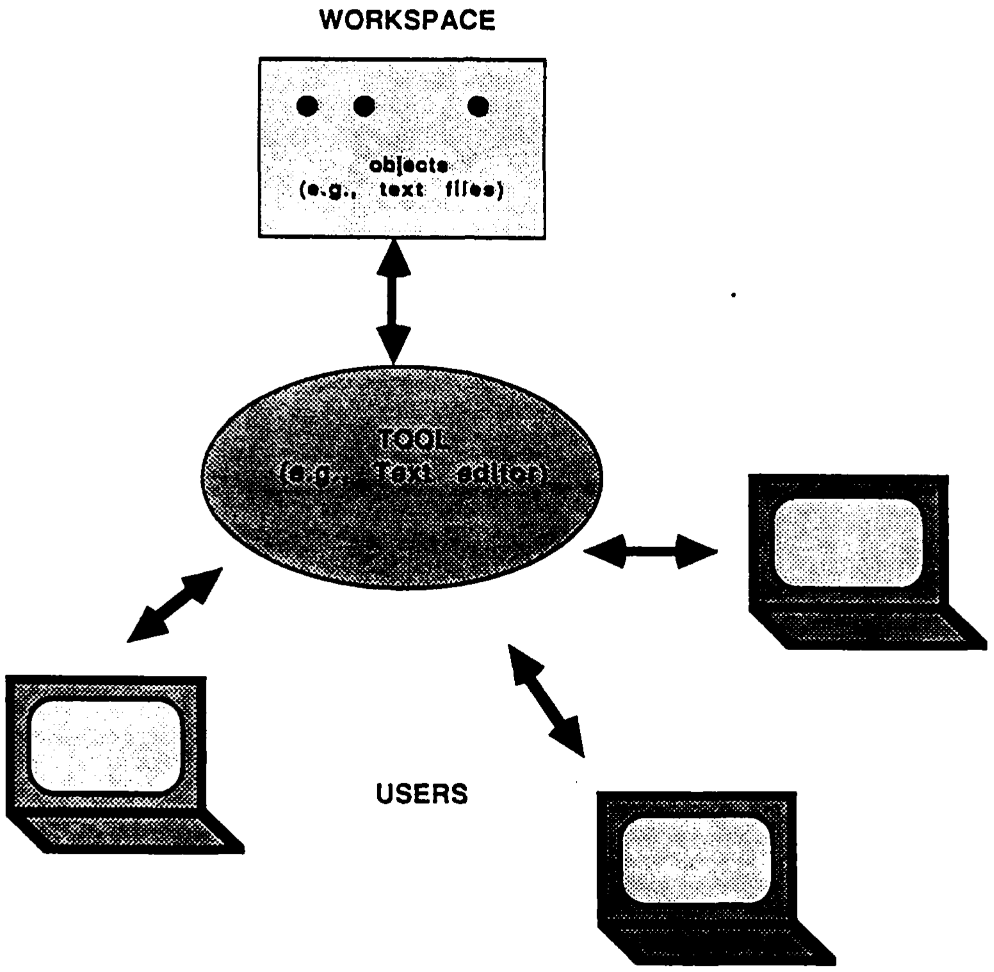


*Fig. 1: General view of the system*

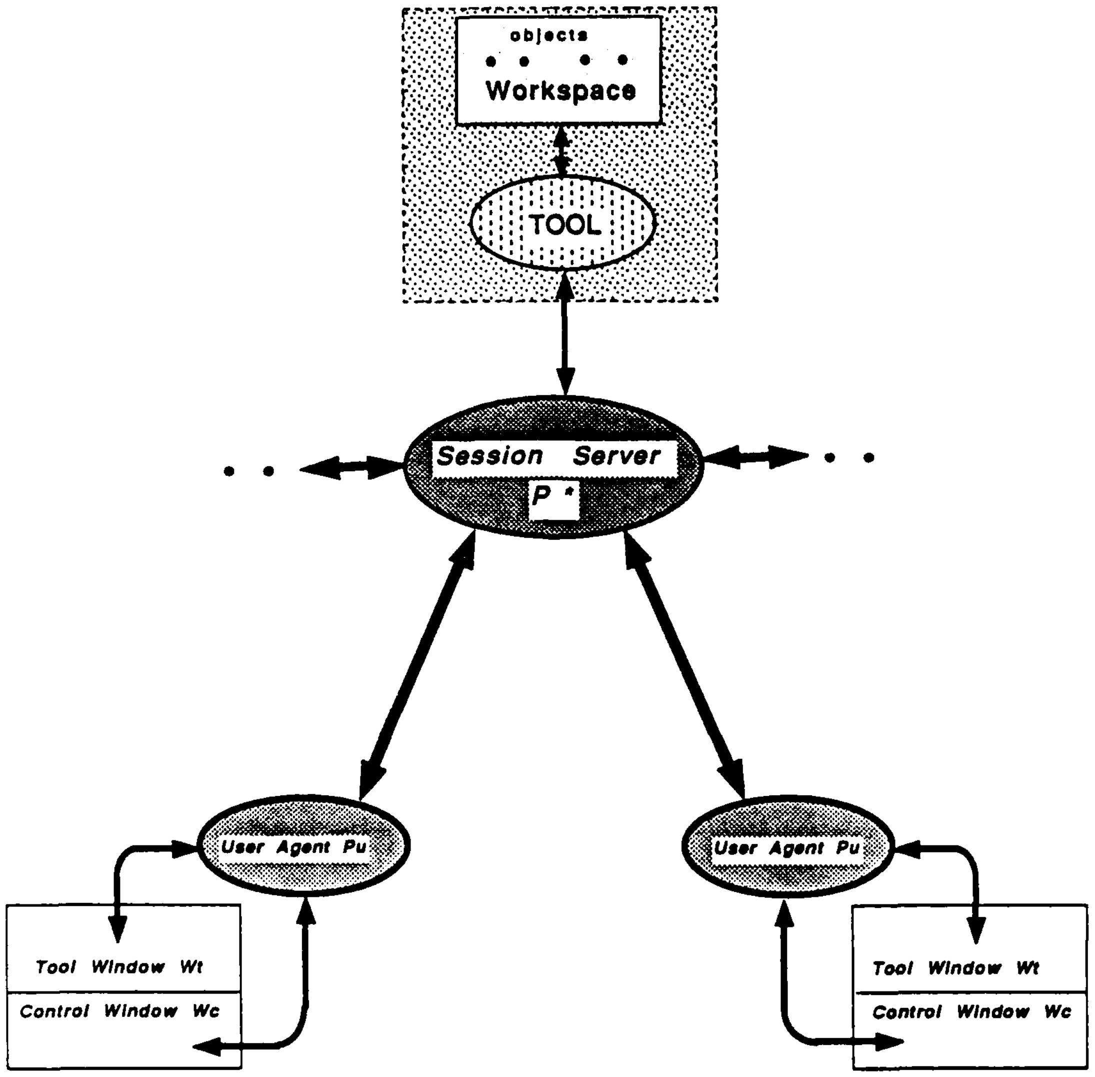


*Fig. 2: Centeral model*

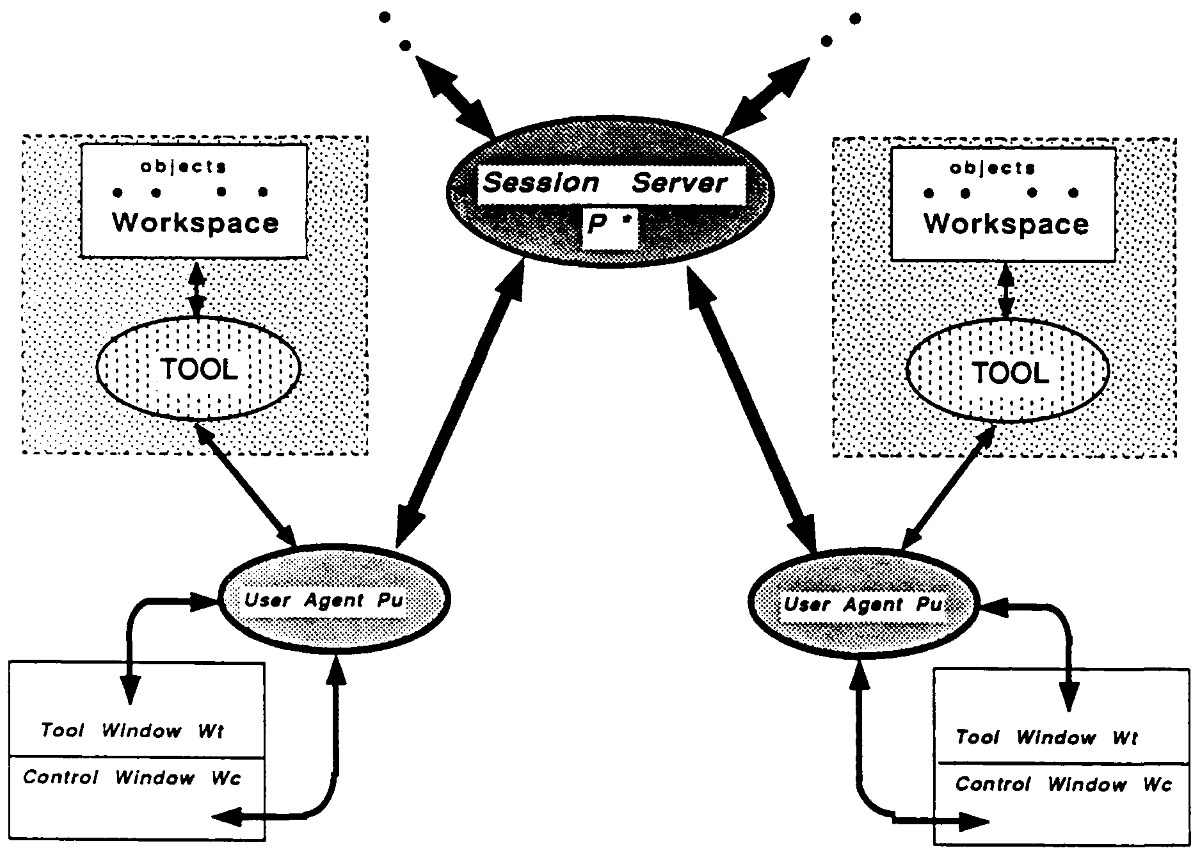


*Fig. 3: Replicated model*

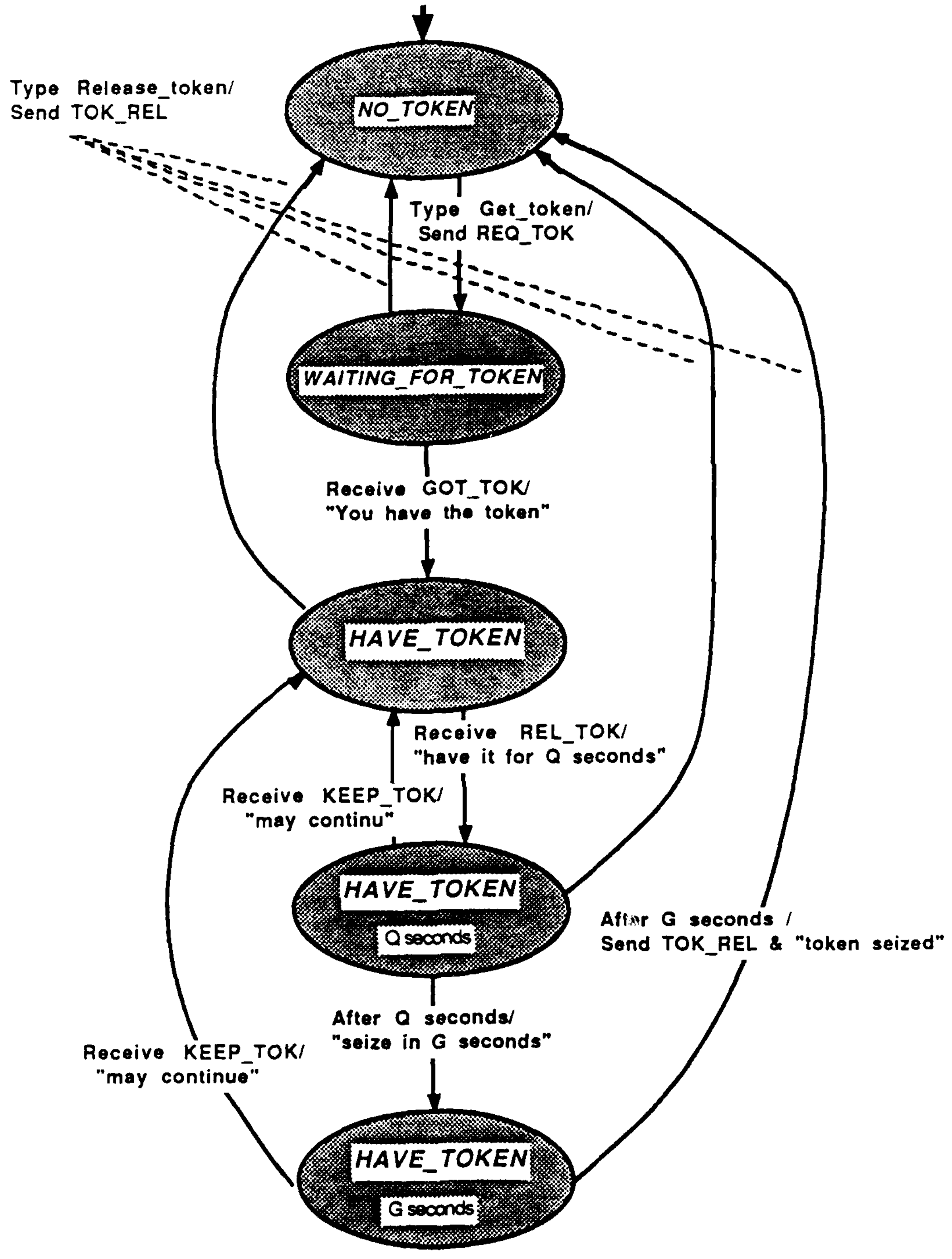
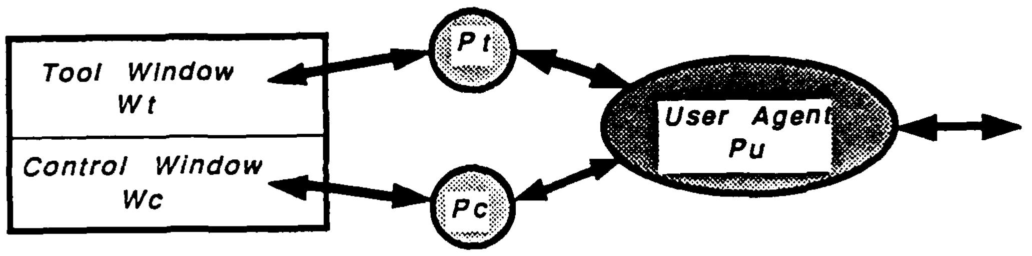
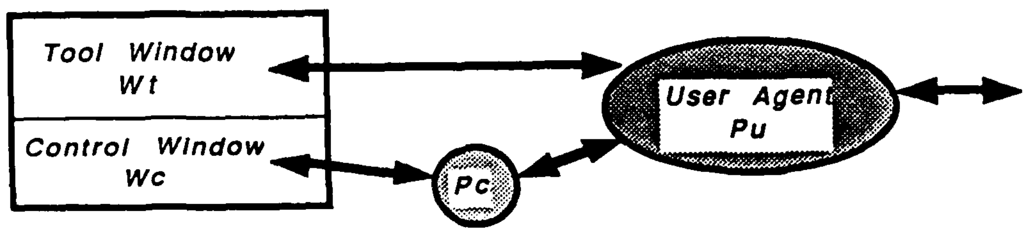


*Fig. 4: Token control states for user agent process*



(a)



(b)

*Figure 5: Connecting windows to user agent process*