

A Distributed and Policy-Free General-Purpose Shared Window System

Thomas Gutekunst, *Student Member, IEEE*, Daniel Bauer, Germano Caronni, *Affiliate Member, IEEE*, Hasan, and Bernhard Plattner, *Member, IEEE*

Abstract— Shared window systems allow collaboration-transparent, single-user applications to be displayed and interacted with on multiple users' workstations, enabling the members of a cooperative ensemble to simultaneously share and revise information. This paper presents a system capable of sharing applications running under the X Window System. In contrast to previously implemented systems, our shared window system addresses issues that are crucial for general-purpose use.

Our shared window system is policy-free, i.e. there are no preferred policies for handling issues such as admission and floor control. Instead, it offers a set of essential mechanisms on top of which various policies and user paradigms may be realized. Further, the system distributes the sharing functionality among all sites involved in a cooperative activity. Measurements have shown a positive impact of this on the overall performance of the system and thus justified the viability of the design decisions taken.

I. INTRODUCTION AND MOTIVATION

Today, the technical environment for high-speed data interchange enables people around the world to interact with each other without a need for travelling. They may attend virtual meetings without even leaving their office, being able to retrieve information from their personal or corporate information systems.

Several desktop conferencing systems designed for various purposes have been brought to the market. While the actual needs for desktop conferencing are still being discovered, it is already apparent that the support provided for cooperative work should aim at supporting self-organization of cooperative ensembles instead of disrupting cooperative work by computerizing formal procedures. Members of a cooperating ensemble should be allowed to interact freely, i.e. without being constrained by prescribed procedures or established conversational conventions, through the provision of facilities enabling them to cooperate via joint construction of a common information space [34].

One fundamental facility is a shared visual space, which allows the members of a cooperative ensemble, each on his/her own computer workstation, to simultaneously share and revise information. A possible solution is to build a new set of collaboration-aware applications that explicitly support this facility. Though representing the emerging generation of CSCW appli-

cations, such an approach has several problems. Perhaps the most critical of these is that users are limited to the use of special-purpose collaboration-aware applications. Considering the diversity of existing computer applications, this requirement appears very limiting. Further, the lack of supporting infrastructure requires most collaboration-aware applications to be constructed from scratch.

Shared window systems are another solution to providing a shared visual space. They exploit properties of the base window system to allow joint usage of unmodified applications. Such an approach has several advantages. Firstly, users are not required to use new applications—they can share the applications already in place. Secondly, the system does not need to be modified to support new applications or changes to existing applications. Finally, the task of developing a cooperative environment is greatly reduced. Instead of reimplementing many existing applications, the developers only have to implement a shared window system.

Many desktop conferencing systems offer application sharing capabilities for the X Window System [33]. X is a network-transparent, device-independent windowing and graphics system currently supported by most leading workstation manufacturers. With these desktop conferencing systems, quite a number of shared window systems have been built [1, 16, 4, 5, 9, 11, 18, 23, 27, 24, 28, 29]. From the user's point of view, the main difference between these systems lies in the user interface paradigms and the applicable policies. Implementing a system that allows X applications to be shared, however, is very complex in detail due to some decisions taken when the X protocol was designed [2, 30]. It is therefore reasonable to have just a single general-purpose shared window system that can be used for several desktop conferencing systems.

This paper presents a fully-distributed and policy-free system for sharing applications running under the X Window System. In contrast to previously implemented systems, our shared window system addresses issues that are crucial for general-purpose use. The system is policy-free, i.e. there are no preferred policies for handling issues such as admission and floor control. Instead, it offers a set of essential mechanisms on top of which various policies and user paradigms may be realized. Furthermore, the system distributes the sharing functionality among all sites involved in a cooperative activity which has shown a positive impact on the overall performance of the system.

The remainder of this paper is organized as follows: Section II deals with the requirements that are fundamental to a shared window system. The major design issues, the control mecha-

Manuscript received July 27, 1994, revised October 6, 1994; accepted by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Hannes P. Lubich.

The authors are with the Swiss Federal Institute of Technology (ETH Zurich), Computer Engineering and Networks Laboratory (TIK), 8092 Zurich, Switzerland.

IEEE Log Number 9401230.

nisms, and the system architecture are described in Sections III, IV, and V, respectively. Section VI sketches two rather different applications of our shared window system. An evaluation of the system including performance measurements is given in Section VII. Finally, Section VIII concludes the paper.

II. FUNDAMENTAL ISSUES

In order to provide a generic service, a shared window system must address a number of issues. This section deals with the issues which we believe to be crucial. Some of these issues have already been discussed previously. However, as far as we know, there is currently no shared window system that addresses all of them.

2.1. Background

Schmidt and Bannon [34] take the term “cooperative work” as “the general and neutral designation of multiple persons working together to produce a product or service”. In cooperative work, the sharing of information plays an essential role in that it provides a “shared context” for the cooperation [31]. A shared context is a set of objects where the objects as well as the actions performed on the objects are visible to a set of users [15].

Ellis et al. [15] define “groupware” as “computer-based systems that support groups of people engaged in a common task (or goal) and that provide an interface to a shared environment”. Thus, groupware denotes systems that address the nature of cooperative work. A number of groupware systems have implemented the notion of a “shared view”, where multiple users perceive the same object in the same state and perceive any changes in the state of the object concurrently. Any changes to the object by one user will immediately be perceivable to the other users [34]. Shared window systems exactly support the concept of shared views in that they permit sharing views and interactions with single-user applications. Greenberg [20] mentions several possibilities how sharing of single-user applications can significantly augment people’s ability to work together.

Sharing single-user applications is one approach for constructing multi-user interfaces. Another approach is to develop special-purpose applications that explicitly handle the collaborative situation. Lauwers and Lantz [24] identify the two approaches as “collaboration transparency” and “collaboration awareness”, respectively. Ahuja et al. [3] refer to systems that support collaboration transparency as “open systems” while collaboration-aware systems are termed “closed systems”. Many closed systems—database systems in particular—insulate users from each other and provide the illusion of being the only user of the system [15, 31]. In cooperative work, however, users wish to be aware of other users’ activities.

Developers of collaboration-aware multi-user applications are confronted with a series of challenging problems [21]. Developers’ experience is still heavily based on single-user applications. That is why many groupware systems fail in getting accepted by the end users. As a general recommendation, Grudin [21] suggests that already successful single-user applications should be extended for the use in a group context by

adding groupware features. This fits well with the notion of application sharing, which makes existing single-user applications usable in a group context.

2.2. Sharing Metaphor

A shared window system allows collaboration-transparent, single-user applications to be displayed and interacted with on multiple users’ workstations simultaneously. The terms “collaboration-transparent” and “single-user” denote that the applications were actually constructed for a single-user environment and hence are not aware of being run in a group context.

A principal requirement to a shared window system is that applications need not be modified in order to be sharable. This allows even “off-the-shelf” applications to be shared. The possibility to share applications to which the users are already familiar is very important since having to learn new interfaces for sporadic tasks discourages many users [11].

A shared window system has to be transparent for the applications that are to be shared, i.e. the applications should not see any difference between the shared window system and the base window system. Also, users should be able to run applications under the shared window system without taking special preparations. Even more important, the run-time behavior of applications should not be affected by the shared window system, neither in quality nor in performance. As for the latter, user interactions with applications running under a shared window system are slightly slower than those running under the base window system due to the overhead caused by the shared window system. However, this delay should not be noticeable to the user.

2.3. Relaxed WYSIWIS

“WYSIWIS” stands for “What You See Is What I See” and denotes interfaces in which the shared context is guaranteed to appear the same to each user [15]. Stefik et al. [35] define an interface to be “strictly WYSIWIS” when all users see exactly the same as well as where the other users are pointing. In practice, strict WYSIWIS was found to be too limiting. The shared focus of strict WYSIWIS makes concurrent operation with different applications by different users impossible.

With “relaxed WYSIWIS”, only portions of the screen are shared by distinguishing shared windows from private windows [8, 15, 24, 35]. Shared windows are visible to each user while private windows are displayed locally only. Furthermore, relaxed WYSIWIS allows the users to rearrange private as well as shared windows as desired and to pursue independent activities on their workstation.

Shared window systems are based on the notion of shared windows that are visible on each users’ workstation. Changes in the contents of a shared window are immediately reflected in all users’ shared window instances. Relaxed WYSIWIS distinguishes shared window systems from display sharing systems where the whole screen content is replicated to all users in a conference [17]. However, as far as the contents of shared windows is concerned, shared window systems strictly adhere to the WYSIWIS concept.

Even relaxed WYSIWIS may be too restrictive, especially when users have widely differing roles, knowledge, and abili-

ties [20]. Shared window systems cannot provide personalized views. This limitation is fundamental and can only be overcome with collaboration-aware applications.

2.4. Activity Awareness

To achieve successful cooperation, there is a clear requirement that members of a cooperative ensemble are aware of individual and group activities [14, 15, 31, 35]. Dourish and Bellotti [14] define “awareness” as the “understanding of the activities of others, which provides a context for your own activity”.

In cooperative work, users wish to be aware of what other users are doing. This requires the propagation of each user’s activity to the other users [31]. There are several mechanisms for this propagation. Dourish and Bellotti [14] distinguish informational and role-restrictive approaches from the approach to present “shared feedback”, which makes information about individual activities apparent to the other users by presenting feedback on operations within the shared workspace. Shared feedback resembles the mechanisms that cooperative ensembles already know from natural cooperation and allows users to vary their activities dynamically in response to the changing state of affairs [14].

By their nature, shared window systems provide shared feedback and achieve awareness for the collaborative situation—changes in the contents of a shared window are immediately reflected in all users’ shared window instances.

2.5. Workspace Management

The workspace management is responsible for visualizing cooperative work in an appropriate way. A user wants to know which items are private and which are subject of cooperation. Furthermore, it is important that the user is able to associate shared items to a cooperative activity and to understand the relationships among them.

Under a shared window system, a user should be able to distinguish private windows from shared ones [24]. For shared windows, the user should also be able to see with whom they are shared or from which user they come from, respectively, and whether or not he/she may provide input to the underlying applications. This may be achieved by visual cues. Crowley et al. [11] argue that the workspace management should not provide a visible “shared workspace” on the workstation’s display, but rather allow to easily mix and arrange shared and private windows as desired in order to allow the users to pursue independent activities on their workstation.

Finally, a shared window system should allow users to make private windows shared and vice versa [24].

2.6. Floor Control

Single-user applications that run under a shared window system are not aware of being run in a group context. Thus, they do not expect input from multiple users. “Floor control” serves as a concurrency control mechanisms that determines at any given point in time which user is allowed to direct input to an application [15]. The right to generate input is denoted by the “floor”, the user currently allowed to do so is called the “floor holder”.

Crowley et al. [11] separate the concept of floor control into

mechanisms and policy. The floor control mechanisms handle the low-level activities of passing the floor and maintaining a synchronized event stream for all users. The floor control policy comprises a set of rules governing floor control, i.e. determining how users request and are granted the floor. Lauwers and Lantz [24] characterize and discuss floor control policies along three dimensions: the scope of the floor (per conference, application, or window), the number of users that can concurrently hold a floor, and how the floor is passed.

Floor policies may support explicit floor passing, where the floor moves from one user to another by an explicit assignment, as well as implicit floor passing, where the floor is implicitly assigned to a user as soon as he/she generates input events. Implicit floor passing corresponds to the concept of having no floor at all. However, low-level mechanisms within the shared window system then have to take care of avoiding inconsistent input events being sent to the application.

Ellis et al. [15] observed that after a learning period, implicit floor passing is not chaotic but surprisingly useful because a social protocol is used for mediation. The fact that users need not perform an explicit operation to request or grant the floor, constitutes an enormous advantage of implicit floor passing. However, while being reasonable for small groups, it is not sufficient to solely rely on social protocols. This is especially important in high-delay environments [24].

The preferred floor control policy for a given situation is depending on the group task, the size of the group, the politics of the group’s interactions, and the application itself [20]. Since any given policy will not be able to satisfy all groups in all situations, a shared window system should not enforce specific floor control policies. It is better to provide simple mechanisms that allow various floor control policies to be built on top of them.

A fundamental limitation of shared window systems is that they do not allow concurrent, independent input to the same shared application because there is only one input focus per application. As with personalized views, collaboration-aware applications are required to handle this situation. However, this brings new demands on users to stay informed about what other users are doing [35]. They can less easily interpret sudden display changes resulting from others’ actions [15].

2.7. User Management

Schmidt and Bannon [34] point out that membership in cooperative ensembles is not stable and often even not determinable. Many cooperative activities do not occur in the context of scheduled meetings, but rather spontaneously and unplanned. Users cannot always anticipate with whom they will be cooperating, nor which items will be subject of cooperative work [24]. In order to support spontaneous interaction, it must be possible to initiate cooperative activities in a “light-weight” fashion. It is desirable—especially for longer-term cooperation—that users are able to join and leave a cooperative activity at any time [20].

A related aspect is admission control. It determines which users are allowed to participate in a cooperative activity as well as how they join. As with floor control, different policies for handling admission control may be appropriate depending on a given setup. As cooperative ensembles typically intersect [34], a

user should also be able to participate in multiple cooperative activities simultaneously.

Dynamic user participation is a key requirement for a shared window system since cooperative activities often evolve in unexpected ways. A shared window system, therefore, should provide mechanisms that support dynamic user participation and allow various admission policies to be realized.

2.8. Conversational Support

Providing only a shared visual space is not meaningful enough for carrying out desktop conferences. The observation that people coordinate their activities via their conversation brings the demand for additional support [15].

Voice conversation allows the users to discuss the contents of shared windows and to coordinate their work. Computer-based communication facilities such as desktop videophones may be used for that purpose. Often, applications are shared between two persons only. In this case, a telephone call may suffice.

In many desktop conferencing systems, we also find telepointing facilities that provide pointing tools visible in all instances of the same shared window. They have proven useful to draw attention to a particular object within a shared window [20, 24, 32]. As with telepointing facilities that serve gesturing purposes, graphical annotation facilities may also be useful in many situations [24, 32]. Graphical annotation enables users to draw on top of shared windows in a way that is transparent to the underlying applications.

In our opinion, a shared window system should not offer telepointing and annotation facilities directly, but rather provide mechanisms that support the construction of such facilities. As with admission and floor control, the reason is that the shared window system should not impose a specific policy.

III. MAJOR DESIGN ISSUES

3.1. Single-Execution vs. Replicated Execution

A primary design decision to be taken was whether to choose a replicated or a centralized architecture with respect to application execution. In a centralized architecture (Figure 1a), a single instance of the application executes. The shared window system

then distributes output to and collects input from each conference participant. [1, 4, 5, 9, 18, 23, 27, 28, 29] are examples for such an architecture.

In a replicated architecture (Figure 1b), an instance of the application executes at each conference site. Here, the shared window system distributes only user input to all the other application instances ensuring that each receives the same sequence of input events. As all instances see the same input, their states remain synchronized, yielding identical output on each site. [16, 11, 24] describe conferencing systems based on a replicated architecture.

Compared to the centralized approach, the replicated approach tends to offer superior response time and reduced network load as only input events are sent across the transport system. Also, it is easier to accommodate differences in display hardware since each application instance can tailor itself to the local characteristics.

However, these advantages disappear when considering the serious synchronization and consistency problems associated with application replication [3, 25]. First of all, the replicated approach does not work if an application to be shared is not available at all sites or if the installed versions are different. Further, it is almost impossible to fulfill the dynamic participation requirement and thus allow for spontaneous interaction.

In a replicated architecture, the shared window system guarantees that the same user input is delivered to all application instances, but it cannot assure the equivalence of input originating from other sources such as data read from files, values of environment variables, or messages from other applications. The general need for data replication constitutes an enormous disadvantage of the replicated architecture.

A related problem is output consistency. Applications may send output to various destinations in the environment, e.g. by writing to a file, sending a document to a printer, or invoking a mailer. When multiple instances of a shared application produce such output, the shared window system should maintain the single-execution semantics. However, this can hardly be achieved without modification of the applications to be shared.

All these arguments considered, we decided to choose the single-execution approach, where applications execute only once. Given the high-speed networks that recently have emerged and the asynchronous nature of the X Network Proto-

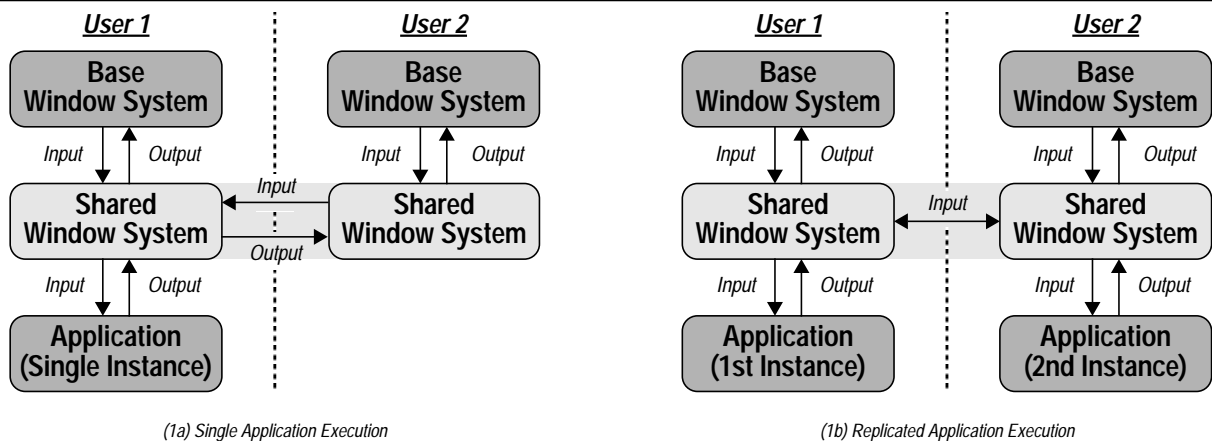


Fig. 1. Single-Execution vs. Replicated Execution

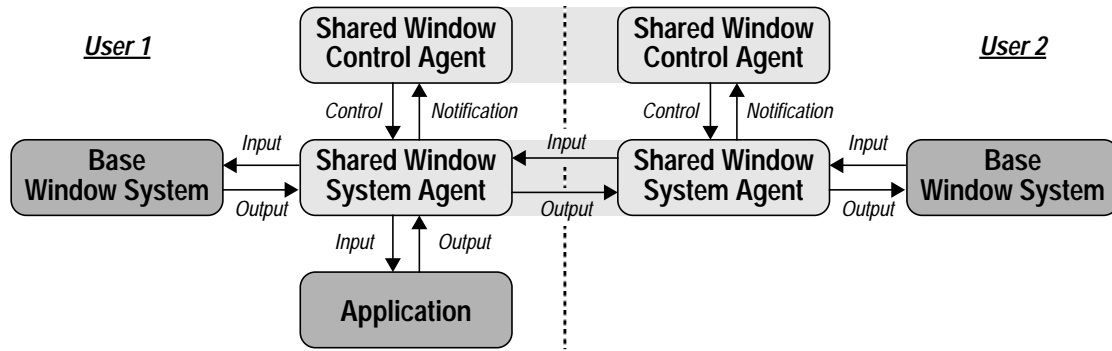


Fig. 2. Separation of Shared Window System from Shared Window Control

col [30], we do not take the distribution of application output as a significant disadvantage. Ahuja et al. [3] report that the single-execution approach does not suffer a significant performance loss as compared to the replicated approach. Finally, it is also worth mentioning that the single-execution approach saves license costs for applications to be shared.

3.2. Separation of Policies from Mechanisms

Our approach strictly separates policies from the underlying mechanisms. This allows various policies and user paradigms to be realized on top of a set of essential mechanisms such that a wide range of cooperatives styles can be supported [31, 32]. Consequently, we use a two-layer model (Figure 2): application sharing is carried out by cooperating “Shared Window System Agents” (system agents), which offer the mechanisms, whereas user paradigms and policies are implemented by cooperating “Shared Window Control Agents” (control agents). The mechanisms are described in Section IV.

A control agent is the initiator of all sharing activities. On a per-application basis, it essentially turns sharing on and off, and implements admission and floor control. Admission control is used to allow or deny application sharing for specific applications and/or users. Another important functionality of the control agent is the administration of users and hosts. Since the system agent does not know anything about users, this information is stored and handled by the control agent.

By separating the shared window system from shared win-

ow control, it is easy to replace one control agent by another such that simple X sharing systems as well as complex desktop conferencing systems can be constructed on top of our shared window system. Section VI briefly describes the respective control agents for (1) “EasyShare”, a light-weight sharing system suitable for a user help desk, and for (2) “JVTOS” (Joint Viewing and Tele-Operation Service), a complex conferencing system comprising application sharing, telepointing, and audio/video conferencing [12, 13, 22].

3.3. Centralized vs. Distributed Architecture

The choice of an appropriate topology is also an issue of major impact. With a centralized architecture (Figure 3a), all conference activities are mediated by a central conference server. In contrast, a distributed architecture (Figure 3b) distributes the sharing functionality among all sites that are involved in a conference. For several reasons, we decided to choose a distributed architecture.

First of all, a centralized architecture is much more vulnerable to failures of either the conference server itself or the communication links connecting to it. Further, the load of the conference server grows as the number of participants and activities increases. This is an obvious obstacle for scalability. Finally, all conference participants but the one running the conference server suffer from a long communication path which results in unnecessarily bad performance for user interaction with locally executing applications.

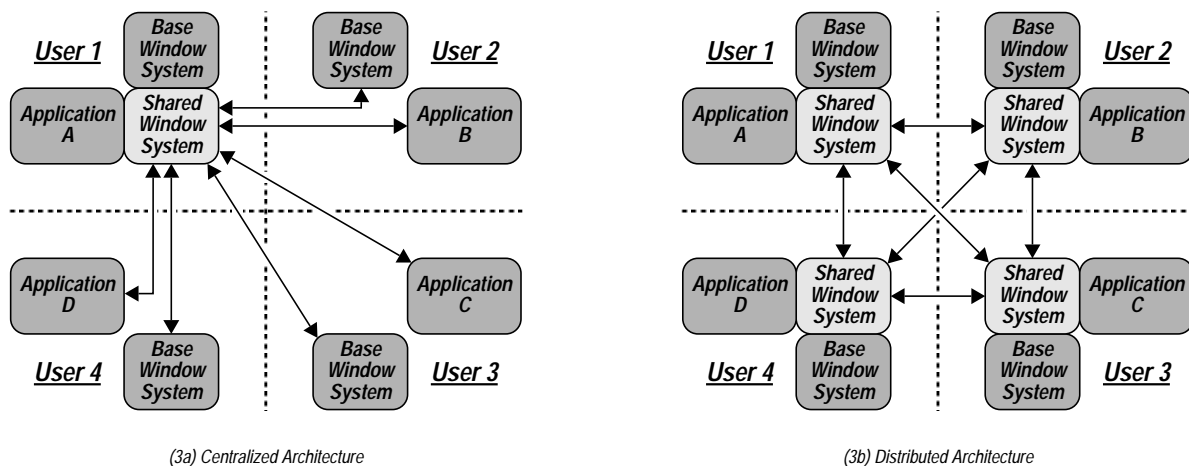


Fig. 3. Centralized vs. Distributed Architecture

The master/slave concept inherent in the centralized architecture hampers user autonomy. It is our strong belief that the concept of independent cooperating entities is more promising for the further development of open systems for cooperative work. Note that the implementation of a chaired conference paradigm is still possible on the layer of shared window control.

IV. CONTROL MECHANISMS

This section describes the mechanisms provided by the core of our shared window system. A control agent locally controls the shared window system by communicating with the system agent over a well-defined “Shared Window Control Protocol” (control protocol). It provides means for:

- admission control (adding/removing users to/from applications),
- floor control (controlling user input to applications),
- application information,
- miscellaneous support.

Figure 4 shows the elements of the control protocol. The shared window system is controlled by issuing requests whereas notifications are accomplished via events. Requests to which an immediate reply is given are referred to as round-trip requests.

4.1. Admission Control

Many desktop conferencing systems offer admission policies comprising primitives such as INVITE, JOIN, DROP, and LEAVE. In our shared window system, there is no preferred admission policy. Instead, the shared window system is flexible in

that it offers the concept of an “application owner” and the concept of attaching/detaching additional users to/from applications.

Each application running in the context of the shared window system has an application owner, which is defined as the user on whose display the application is displayed when not being shared. Usually, this corresponds to the user that launched the application. An application owner ultimately decides about admission of other users to his/her applications. Thus, no user can access other users’ applications without granted permission by the application owner!

Attaching a user to an application means sharing the application with an additional user, which will get a duplicate of the application windows on his/her display. Depending on the current floor control policy, the attached user might also provide input to the application. Detaching disassociates an attached user from an application, i.e. the application is no longer shared with that user. The attach operation is valid for the application owner’s control agent only while the detach operation may be issued by the control agent of the application owner as well as of the user in question.

The fact that not all control agents are allowed to use the attach/detach mechanisms does not violate the concept of a policy-free shared window system since cooperating control agents may realize any admission policy.

With the application owner and the attach/detach concept, any admission policy can be implemented by cooperating control agents.

Functionality Class	Request Type
Admission Control	AttachUser
	DetachUser
	GetUserGroup *
Floor Control	EnableInput
	DisableInput
	SetInactivityTimeout
	SetInputSensitivityMask
	GetInputGroup *
	GetFloorHolder *
Application Information	ListApplications *
	GetApplicationName *
	GetApplicationOwner *
Miscellaneous Support	SendMessage
	BroadcastMessage
	SetXResourceMask
	SetXEventMask
	CreateControlWindow
	ChangeControlWindow
	DestroyControlWindow

(4a) Control Elements

*) Round-trip request (i.e. request/reply pair)

Functionality Class	Event Type
Admission Control	UserAttached
	UserDetached
Floor Control	InputEnabled
	InputDisabled
	FloorShifted
	FloorReceived
Application Information	FloorLost
	ApplicationStarted
Miscellaneous Support	ApplicationTerminated
	MessageReceived
	XResourceCreated
	XResourceDestroyed
	XEventReceived

(4b) Notification Elements

Fig. 4. Elements of the Shared Window Control Protocol

4.2. Floor Control

As our shared window system is policy-free, it does not support any specific floor control policy. It offers a set of floor control mechanisms only, that handle the low-level activities of passing the floor token and maintaining a synchronized event stream for all attached users.

Floor control is applied on a per-application basis. The application owner's control agent grants the floor to a user and also revokes it using `EnableInput` and `DisableInput` requests. Our shared window system allows for more than one user to be floor holder simultaneously. These users are referred to as "candidate floor holders". Inside the shared window system, a scheme called "implicit floor passing" is applied: the floor token is assigned implicitly as soon as a candidate floor holder generates input events.

While more than one user may be candidate floor holder, there is at most one "current floor holder" at any given point in time. The `SetInactivityTimeout` request allows to specify a minimum interval for input inactivity of the current floor holder that is required before an implicit assignment to another user may take place.

Input events are generated by actions such as pressing a key on the keyboard, pressing a mouse button, moving the mouse into a window, or selecting a window. However, it is not appropriate to implicitly pass the floor on each input event. Practicality may also vary from application to application. Consider `xfig` [36], which is a drawing program. To draw a line, the user first selects a starting point by clicking the mouse button, then moves the mouse to the end point of the line and clicks again. This action must not be interrupted by a floor shift. For this kind of application, shifting the floor on every mouse click is not desirable. Therefore, the implicit floor shift mechanism is configurable on a per-application basis. A control agent may set an input sensitivity mask to select the events that trigger implicit floor passing. By default, the floor is passed on keyboard input and mouse clicks only.

The floor control mechanisms provided by our shared window system are sufficient to implement various floor control policies, as illustrated in Section VI.

4.3. Application Information

Control agents are automatically notified about start and termination of each application that is running locally under the shared window system. Further, it is possible to request a list of the currently running local applications as well as of the applications to which the local user is attached. Application information comprises a globally valid application identification, its name (a text string), and the identification of the application owner.

4.4. Miscellaneous Support

Our shared window system supports communication between control agents. The control protocol allows a control agent to send messages to other control agents. The addressing scheme is quite simple. A message can be sent either to a single user's control agent (`SendMessage`) or to the control agents of all users attached to a given application (`BroadcastMessage`).

In order to support more complex conferencing facilities, the

shared window system provides application-related resource and event information. Control agents can select the X resource types in which they are interested (`SetXResourceMask`). The control agent will then be notified whenever a shared application creates or destroys an X resource of selected type. Further, the control agent can set an event mask (`SetXEventMask`) such that the control agent receives a copy of all X events of selected type that are sent to a shared application. Resource and event selection is done per application. As for X resources, the notification contains resource type and resource identifier as well as a global resource identifier which is globally valid within the shared window system with respect to a given application. The global identifier allows several users' instances of the same resource to be related to each other.

Our shared window system also supports enhanced user interfaces by allowing per-application controls to be included into the top-level window of shared applications. The control protocol provides mechanisms for creating, changing, and destroying control windows for a given application. Once created, a control window can be drawn into, resized, mapped, and unmapped by the control agent. The control windows are handled by the shared window system such that neither the shared application nor the window manager are aware of it. Upon resizing of either the application window or the control window, the shared window system automatically rearranges both according to the gravities defined for the control window.

V. IMPLEMENTATION

The environment of our shared window system consists of several interconnected UNIX workstations (Sun Sparcstations), with usually one user per workstation. In order to use the shared window system, a workstation needs to run a system agent, which intercepts all access to the local X server, and a control agent. Both agents are automatically initiated upon start-up of the user's workstation environment in an analogous way as the X server and the X window manager are started.

The commonly used approach for sharing X applications is to construct an X multiplexer [6] that intercepts the X display connection between an application (the X client) and the associated X display server and then multiplexes the X protocol data stream [30]. The X multiplexer distributes application output (X requests) to and collects input (X replies, events, and errors) from the users among which the application is shared. As the X Window System was not designed for application sharing purposes, there are many problems an X multiplexer has to cope with [2]. In particular, it must convert resource identifiers, atom numbers, window coordinates, pixel values, key codes, and sequence numbers as well as provide fallbacks to support X servers with different capabilities.

Our shared window system is based on a distributed approach. The functionality of the X multiplexer is split up into a "pseudo server" near the X client and "pseudo clients" near the X servers (Figure 5). The pseudo server communicates with the pseudo clients using a proprietary extension to the X protocol that we here refer to as X'. The X protocol cannot be used in a multicast environment. Therefore, we designed the protocol extension X' in a way that allows multicast to be used for distrib-

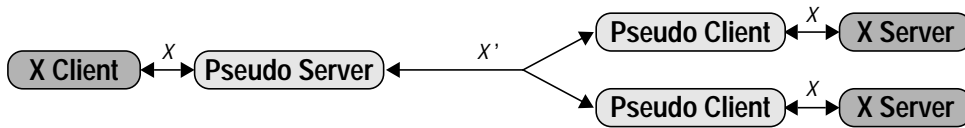


Fig. 5. Distribution of the X Multiplexer into Pseudo Server and Pseudo Clients

uting X requests—an approach already proposed by Ahuja et al. [3]. This fits well with emerging high-speed networks. The main advantage of the multicast capability is that it saves bandwidth and thus makes even conferences with a large number of participants feasible.

The pseudo server multiplexes X protocol data streams and keeps track of X resources created by the applications. For each user, a pseudo client maps the X protocol data streams to the respective local characteristics. Keeping track of X resources enables the shared window system to attach additional users to applications and thus allows for dynamic user participation. Further, replies to many round-trip requests can be generated by the local pseudo server, which results in low latency.

A Shared Window System Agent consists of a pseudo server that multiplexes applications and of a pseudo client that handles applications to which the local user is attached. With the shared window system being active, applications do not connect to the local X server, but to the local system agent. This is achieved by setting the environment variable DISPLAY accordingly. The system agent creates an entry in its tables for that application and then establishes a connection to the local X server. As long

as the application is not shared with other users, the system agent does nothing but forwarding X PDUs and keeping track of resources (Figure 6). This task must not be time consuming since applications are interacted with locally and not shared most of the time. For resource tracking, we adapted an efficient algorithm proposed by Chung et al. [10] that does not consume much storage capacity.

When a user is to be attached to the application, the new user's X server is brought into the current state required for the application to run, i.e. all resources kept track of are created on the new X server. Then, multiplexing and filtering are enabled. The local system agent sends X requests to the attached user's system agent and receives X replies, events, and errors from the remote system agent according to the current floor control settings. The remote system agent maps the X PDUs to/from local characteristics. Figure 7 depicts a scenario where two users both share an application with one another.

Low-level communication issues are separated from the shared window system core such that the shared window system can easily be ported to other platforms. A communication submodule provides a channel abstraction between cooperating entities that is independent of the underlying communication mechanisms. Currently, we support the mechanisms provided by a typical UNIX kernel: internet and UNIX-domain sockets, and shared memory. Among these, the communication submodule automatically selects the most efficient communication mechanism. Other transport systems and protocols (e.g. N-ISDN, ATM, multicast protocols) may be supported in the near future without any changes to the shared window system core. So-called "group channels" allow data to be sent to multiple entities at the same time without having to worry about using multicast packets or employing other mechanisms.

VI. APPLICATION SCENARIOS AND EXPERIENCE

This section describes the respective control agent instantia-

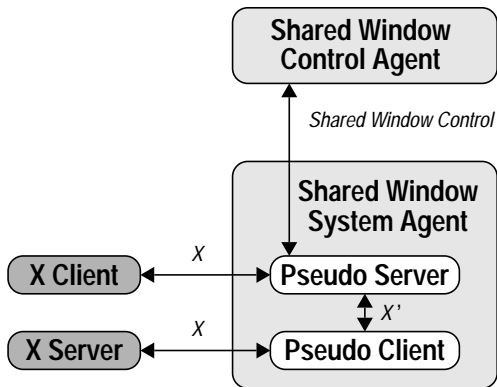


Fig. 6. Running an Application under the Shared Window System

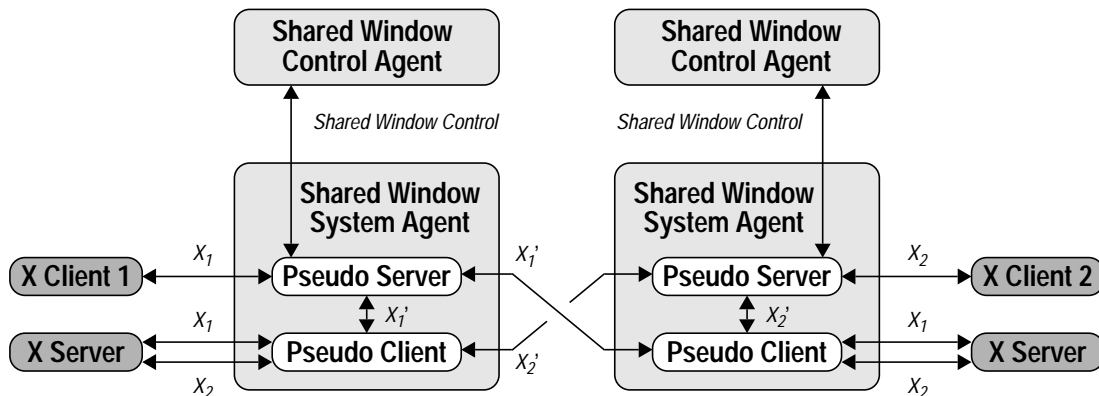


Fig. 7. Two Users both Sharing an Application with one another

tions for (1) “EasyShare”, a light-weight sharing system suitable for spontaneous application sharing, and for (2) “JVTOS” (Joint Viewing and Tele-Operation Service), a complex conferencing system comprising application sharing, telepointing, and audio/video conferencing.

6.1. EasyShare / User Help Desk

EasyShare is simple and easy to handle while providing all required functionality for daily use. It was mainly built for spontaneous application sharing where applications are typically shared for a short period of time only. The idea behind EasyShare is a simple export/import paradigm. An application which is being shared is said to be exported by the application owner and imported by other users.

Among the applications executing locally, each user may specify which applications are to be exported as well as to whom. Those users can accept or refuse attachment to these applications by selecting the applications to be imported. As only the owner can grant admission to an application, unwanted sharing of applications is impossible.

The floor control policy in EasyShare is simple and straightforward. The application owner assigns and withdraws the floor to/from the attached users. These may also issue floor requests themselves which, however, will not be followed unless the application owner confirms such requests. As the floor may be assigned to users without withdrawing it from others, multiple users may hold the floor simultaneously such that they may type concurrently.

EasyShare was implemented on top of our shared window system. Since the shared window system does not provide policies for admission control and floor control, the desired functionality was implemented within the control agents. Admission control requires that EasyShare control agents communicate with each other. Import and export requests are sent with the `SendMessage` and `BroadcastMessage` requests provided by the shared window system, as are the replies. The floor control policy was implemented in an analogous way.

The principal application of EasyShare is our user help desk. When a users gets stuck while working with an application, he/she may call the user help desk. The application is then shared between the consultant and the user. Conversation takes place through the telephone. The consultant guides the user and at the same time sees what the user actually does.

We use EasyShare for the work with our students. They can call us when they have problems with their exercises. Since we cannot only see the application but also actively take part, problems are usually solved much faster than via telephone or electronic mail.

6.2. JVTOS

JVTOS (Joint Viewing and Tele-Operation Service) allows for multimedia collaboration in a heterogeneous workstation environment [12, 13, 22]. JVTOS is issue of work package 4.2 within RACE II project CIO (R2060) [7] and is still under development. Unlike the export/import paradigm of EasyShare, JVTOS uses a session-based model. Sessions are the frame in which cooperation takes place. Besides application sharing, JVTOS provides a telepointing facility for gesturing and a picture-

phone that allows the session participants to communicate audiovisually to each other.

Each session is managed by a chairperson who may invite and drop users. Participants may themselves join and leave sessions. JVTOS offers three admission policies:

- *Closed session:*
Only invited participants are admitted.
- *Joinable session:*
Users may join upon request, but the chairperson decides about admission.
- *Open session:*
Anyone may join without confirmation by the chairperson.

For each application, the respective owner is responsible for selecting an appropriate floor control policy. Four policies are available:

- *Chaired mode:*
The application owner assigns the floor to a participant who returns it after use.
- *Baton mode:*
The current floor holder passes the floor to another participant.
- *FIFO queue mode:*
Floor requests are queued. As soon as the current floor holder returns the floor, it is assigned to the participant heading the queue.
- *Implicit mode:*
The floor is assigned implicitly to a participant as soon as he/she generates input events.

As compared to EasyShare, JVTOS uses an enhanced user interface (Figure 8). The top-level window of each shared application is modified in order to display information about the ongoing session. This information consists of application owner, current floor holder, and applicable floor control policy. Further, a pull-down menu allows to issue floor commands.

In JVTOS, any session participant may contribute shared applications to a session. But, as only the respective owner’s control agent can apply admission and floor control to an application, control over the shared applications is distributed. The control agents of all session participants cooperate by exchanging messages to jointly provide a coordinated session management.

Telepointing requires that the identifiers of shared windows are known and that corresponding shared window instances can be related to each other. This is achieved by selecting notification for window creation/destruction and for map/unmap events.

The control window mechanism is used for displaying session information in the top-level window of shared applications. After creating an empty control window via the shared window system, the control agent directly displays control information via an X display connection to the local base window system.

VII. EVALUATION

To evaluate the performance of our shared window system, we applied several tests to the system. These tests were executed on Sun Sparcstations 2 running Sun OS 4.1.3 in an Ethernet

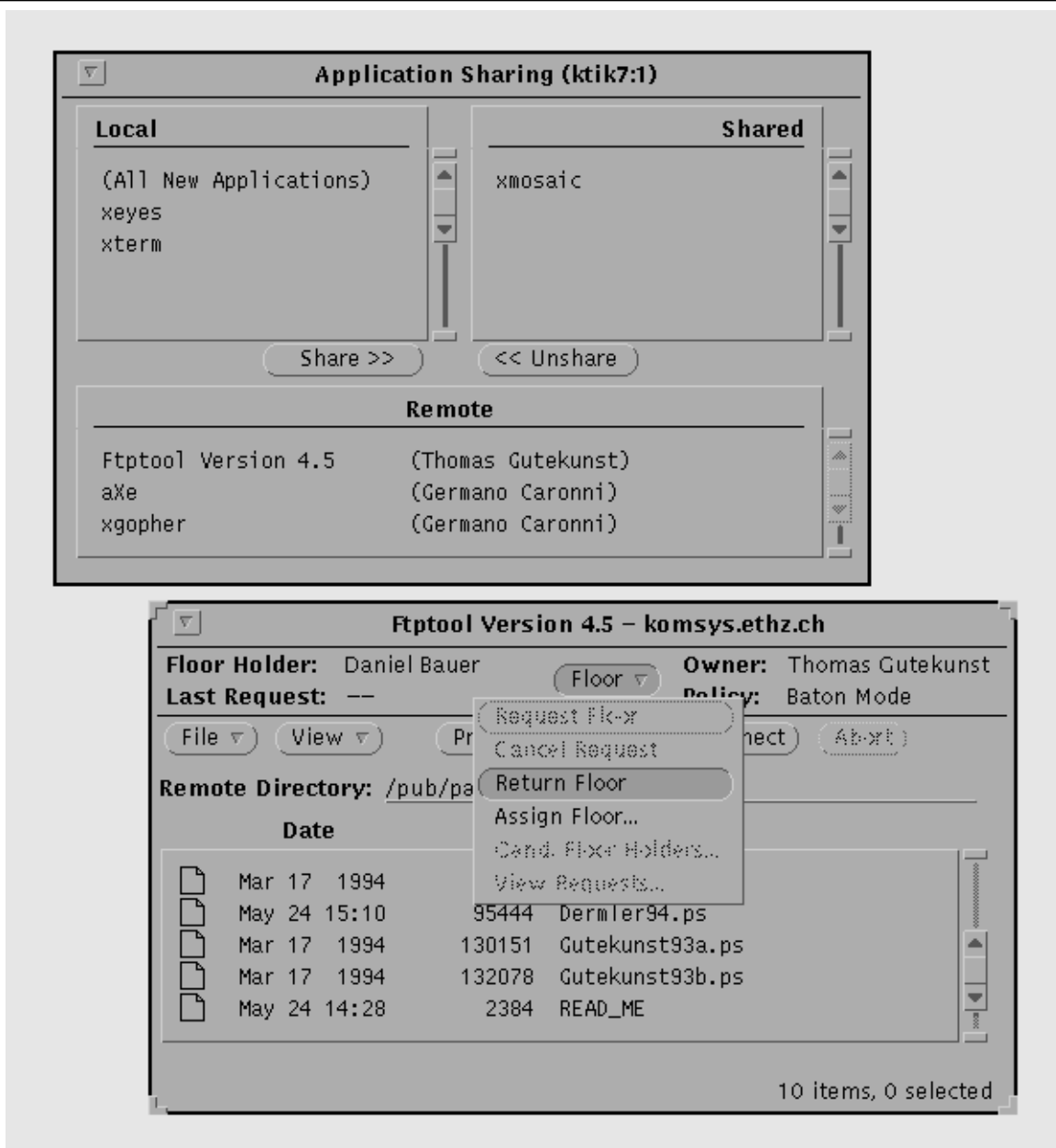


Fig. 8. JVTOS Application Sharing

LAN. The base window system used was the X server that is part of the X11R5 distribution by the MIT X Consortium.

As the shared window system is located on top of the base window system, the performance of the shared window system cannot be better than the performance of the base window system. Therefore, the results must be interpreted in relation to the measurements for the base window system.

The performance tests were selected to answer the following questions:

1. How much does the shared window system degrade application performance in single-user mode when running under the shared window system, as compared to running under the base window system?
2. What characteristics does the shared window system show when scaling the number of users attached to a shared application?

3. Can we gain some performance when running the shared window system on another site than the one running the base window system?

As applications are interacted with locally and not shared most of the time, the relative performance of a shared window system in single-user mode is of great importance. To get an initial figure, we ran the “xbench” test program [19], which computes a so-called “xStones” rating. The relative performance of our shared window system turned out to be 55%.

The question arises how to interpret this figure. In order to see where the shared window system shows good and bad performance respectively, we applied another, more detailed benchmark: the “x11perf” test program [26], which comprises more than 200 test items. Table I summarizes the most interesting results.

As expected, relative performance is better than 55% for

TABLE I
SELECTED RESULTS FROM X11PERF TEST PROGRAM

Operation (x11perf Test Item)	Relative Performance	Operation (x11perf Test Item)	Relative Performance
Dot	41 %	Char in 80-char line (6x13)	50 %
10x10 rectangle	48 %	Char in 60-char line (9x15)	50 %
500x500 rectangle	52 %	Scroll 10x10 pixels	40 %
10x10 rectangle (161x145 stippled)	52 %	Scroll 500x500 pixels	54 %
500x500 rectangle (161x145 stippled)	99 %	Copy 10x10 from pixmap to window	45 %
10-pixel line	52 %	Copy 500x500 from pixmap to window	80 %
500-pixel line	51 %	PutImage 10x10 square	35 %
10-pixel circle	51 %	PutImage 500x500 square	28 %
500-pixel circle	51 %	GetAtomName	25 %
100-pixel wide dashed circle	99 %	GetProperty	25 %
100-pixel wide double-dashed circle	99 %	Hide/expose window via popup (4 kids)	52 %
X protocol NoOperation	9 %	Move window (4 kids)	52 %

items that cause the base window system (i.e. the X server) to perform more complex operations such as for “stippled rectangles” or “dashed circles” (99%). On the other hand, the performance is worse for drawing dots (41%), which is the most simple operation, and even worse for “no operation” (9%). The relative performance is also worse for operations that burden much work to the shared window system such as the PutImage request that requires all pixel values to be converted (28%).

In general, the relative performance of an operation decreases the more its handling is expensive for the shared window system and the less it is expensive for the base window system, respectively. This explains the variation in relative performance figures for different operations.

For round-trip requests such as GetAtomName or GetProperty, a relative performance of 25% is achieved. Round-trip requests are bound by the latency from which a request and its reply suffer as they are sent through the shared window system. With our shared window system, the latency of a round-trip request is 5.4 ms as compared to 1.4 ms with the base window system only. The additional latency caused by a shared window system is of particular importance for highly-interactive applications as it delays application output in reply to user input. However, an additional delay of 4 ms cannot be perceived by a human user.

Obviously, the purpose of shared window systems is to allow more than a single user to interact with an application. Thus, it is also interesting to study the impact of the number of users. Our shared window system was designed such that conferences with even a large number of participants should be feasible. Therefore, we decided to distribute the sharing functionality among a single pseudo server and multiple pseudo clients (one pseudo client per user). Further, the protocol extension X’ allows multicast to be applied for sending packets from the pseudo server to the pseudo clients. Thus, we actually expect the number of attached users to be of negligible impact.

However, the current implementation emulates multicast by multiple unicasting of packets. This causes an additional latency of about 1.4 ms per user due to the system calls required to

multiply send the packets. Figure 9 gives relative performance figures gained from a proprietary test program that measures how long the UNIX “more” command takes to display a 100-page ASCII text file within an “xterm”. The performance decreases from 68% for a single user down to 47% for five users. As expected, the performance curve flattens as the number of users increases.

As the shared window system and the concurrently running base window system compete for CPU and I/O, we wanted to see whether we can gain performance when running the shared window system on another site than the one running the base window system. In a scenario where the shared window system runs on another site, we measured a latency of 6.1 ms, and “xbench” reported a relative performance of 85% for the shared window system. Moving the shared window system to another processor seems to increase the relative performance, but it also creates additional network traffic. The scenario introduces a dependency on the network load as well as on CPU and I/O load of the second site. Further, it makes the management of the shared window system more complex. We have serious doubts whether this effort is worth the potential performance gain.

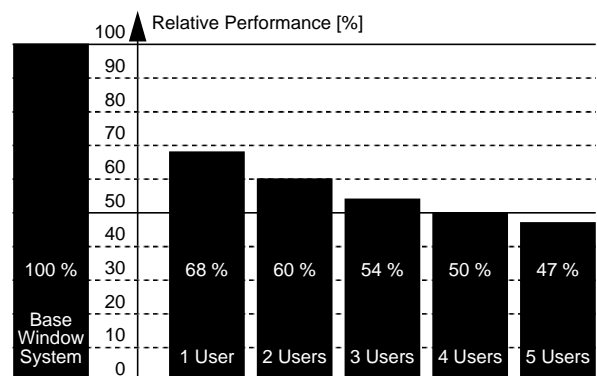


Fig. 9. Impact of the Number of Attached Users

VIII. SUMMARY AND CONCLUSIONS

This paper describes a shared window system capable of sharing applications running under the X Window System. It must be admitted that several such systems have been studied and implemented previously. Similar systems are even available commercially. So what is innovative about our shared window system? The general answer to this question is that our system, in contrast to all shared window systems known to us, addresses fundamental issues that are crucial for a general-purpose shared window system as outlined in Section II.

In particular, our system is policy-free. It separates the core of its shared window system, i.e. cooperating Shared Window System Agents, from the control system, i.e. cooperating Shared Window Control Agents. This separation allows to implement user paradigms and policies independently from the shared window system. Thus, various paradigms can be provided on top of a single shared window system.

Performance measurements have justified the viability of the design decisions taken. The fully-distributed approach, which distributes the load among all workstations involved in a conference, has shown good scalability characteristics. Further, the system is very responsive such that running an application under the shared window system is not significantly less performant than when running under the base window system. Considerable performance improvements have been achieved with our communication submodule that automatically selects the most efficient communication mechanism.

From our experience, the intuitive performance of the shared window system is "fast enough" as not to affect the usability for daily work. As our shared window system is part of JVTOS, which is being developed within RACE II project CIO, the intuitive performance will be judged by a larger community. In particular, it will be interesting to study the impact of large distance to the acceptance of the system when users are confronted with longer transmission delays. We are further looking forward to using the shared window system on a transport system that offers multicast capabilities in order to get experience with large conferences. Initial trials and demonstrations have shown promising results.

It is also worth mentioning that the RACE project "BETEUS" (Broadband Exchange for Trans-European Usage) selected our shared window system for incorporation into a generic platform that supports interactive work between distributed end users. The BETEUS project aims at gaining early experience in real usage of collaborative services in a broadband communications environment. The main focus is on multipoint teleteaching and teleworking.

Our shared window system was written in ANSI-C and implemented on Sun Sparcstations. It is highly portable since the system-dependent parts are isolated in the communication submodule. Thus, the adaption of the submodule is sufficient to successfully port the shared window system to another platform. The shared window system is currently being ported to the Macintosh platform. We are also considering to give the shared window system into public domain in order to promote application sharing on a variety of other workstation types and thus encouraging cooperative work in a heterogeneous environment.

ACKNOWLEDGEMENTS

The work discussed in this paper was performed in the context of the RACE II project CIO (R2060) and specifically within the work package 4.2, in which many of the ideas put forward were born. We would like to acknowledge the contributions of all involved partners that were left unmentioned in this text.

Participation in CIO was financed by the Swiss Confederation under grant no. BBW-R2122.

REFERENCES

- [1] Hussein M. Abdel-Wahab, Mark A. Feit: "XTV: A Framework for Sharing X Window System Clients in Remote Synchronous Collaboration". *Proceedings, IEEE Conference on Communications Software: Communications for Distributed Applications and Systems*, pp. 159 - 167. Chapel Hill, 1991.
- [2] Hussein Abdel-Wahab, Kevin Jeffay: "Issues, Problems and Solutions in Sharing X Clients on Multiple Displays". *Technical Report TR92-042, University of North Carolina*. Chapel Hill, 1992.
- [3] S.R. Ahuja, J.R. Ensor, S.E. Lucco: "A Comparison of Application Sharing Mechanisms in Real-Time Desktop Conferencing Systems". *Proceedings, ACM Conference on Office Information Systems*, pp. 238 - 248. Cambridge (MA), 1990.
- [4] Michael Altenhofen: "Erweiterung eines Fenstersystems für Tutoring-Funktionen". *Diploma Thesis at Universität Karlsruhe*. Karlsruhe, 1990.
- [5] Michael Altenhofen, Jürgen Dittrich, Rainer Hammerschmidt, Thomas Käppler, Carsten Kruschel, Ansgar Kückes, Thomas Steinig: "The BERKOM Multimedia Collaboration Service". *Proceedings, First ACM International Conference on Multimedia*, pp. 457 - 463. Anaheim, 1993.
- [6] John Eric Baldeschwieler, Thomas Gutekunst, Bernhard Plattner: "A Survey of X Protocol Multiplexors". *ACM Computer Communication Review*, Vol. 23, No. 2, pp. 13 - 22. ACM Press, 1993.
- [7] Wulfdieter Bauerfeld: "RACE-Project CIO (R2060): Coordination, Implementation and Operation of Multimedia Tele-Services on Top of a Common Communication Platform". *Proceedings, International Workshop on Advanced Communications and Applications for High Speed Networks (IWACA '92)*, pp. 401 - 405. München, 1992.
- [8] Richard Bentley, Tom Rodden, Pete Sawyer, Ian Sommerville: "Architectural Support for Cooperative Multiuser Interfaces". *IEEE Computer*, Vol. 27, No. 5, pp. 37 - 46. IEEE Computer Society Press, 1994.
- [9] Carsten Bormann, Gero Hoffmann: "Xmc and Xy—Scalable Window Sharing and Mobility, or From X Protocol Multiplexing to X Protocol Multicasting". *Proceedings, 8th Annual X Technical Conference*. Boston, 1994.
- [10] Goopeel Chung, Kevin Jeffay, Hussein Abdel-Wahab: "Accommodating Latecomers in Shared Window Systems". *IEEE Computer*, Vol. 26, No. 1, pp. 72 - 74. IEEE Computer Society Press, 1993.
- [11] Terrence Crowley, Paul Milazzo, Ellie Baker, Harry Forsdick, Raymond Tomlinson: "MMConf: An Infrastructure for Building Shared Multimedia Applications". *Proceedings, ACM 1990 Conference on Computer-Supported Cooperative Work (CSCW '90)*, pp. 329 - 342. Los Angeles, 1990.
- [12] Gabriel Dermier, Konrad Froitzheim: "JVTOS—A Reference Model for a New Multimedia Service". *Proceedings, 4th IFIP Conference on High Performance Networking (hpn '92), paper D3*. Liège, 1992.
- [13] Gabriel Dermier, Thomas Gutekunst, Bernhard Plattner, Edgar Ostrowski, Frank Ruge, Michael Weber: "Constructing a Distributed Multimedia Joint Viewing and Tele-Operation Service for Heterogeneous Workstation Environments". *Proceedings, Fourth IEEE Workshop on Future Trends of Distributed Computing Systems*, pp. 8 - 15. Lisbon, 1993.
- [14] Paul Dourish, Victoria Bellotti: "Awareness and Coordination in Shared Workspaces". *Proceedings, ACM 1992 Conference on Computer-Supported Cooperative Work (CSCW '92)*, pp. 107 - 114. Toronto, 1992.
- [15] C.A. Ellis, S.J. Gibbs, G.L. Rein: "Groupware: Some Issues and Experiences". *Communications of the ACM*, Vol. 34, No. 1, pp. 38 - 58. ACM Press, 1991.
- [16] J.R. Ensor, S.R. Ahuja, D.N. Horn, S.E. Lucco: "The Rapport Multimedia Conferencing System: A Software Overview". *Proceedings, Second IEEE Conference on Computer Workstations*, pp. 52 - 58. Santa Clara, 1988.
- [17] Paul F. Fitzgerald, Nina Y. Rosson, Linda Uljon: "Evaluating alternative display sharing system architectures". *Proceedings, IEEE Conference on Communications Software: Communications for Distributed Applications & Systems*, pp. 145 - 157. Chapel Hill, 1991.
- [18] Daniel Garfinkel, Bruce C. Welti, Thomas W. Yip: "HP SharedX: A Tool for Real-Time Collaboration". *Hewlett-Packard Journal*, Vol. 45, No. 2, pp. 23 - 36. 1994.
- [19] Claus Gittinger: "xbench". Software package in public domain. Copyright 1988, Siemens Munich. Available via anonymous ftp and internet information retrieval services.
- [20] Saul Greenberg: "Sharing views and interactions with single-user applications". *Proceedings, ACM Conference on Office Information Systems*, pp. 227 - 237. Cambridge (MA), 1990.

- [21] Jonathan Grudin: "Groupware and Social Dynamics: Eight Challenges for Developers". *Communications of the ACM, Vol. 37, No. 1, pp. 92 - 105*. ACM Press, 1994.
- [22] Thomas Gutekunst, Thomas Schmidt, Günter Schulze, Jean Schweitzer, Michael Weber: "A Distributed Multimedia Joint Viewing and Tele-Operation Service for Heterogeneous Workstation Environments". *Proceedings, GI/ITG Arbeitstreffen Verteilte Multimedia-Systeme, pp. 145 - 159*. Stuttgart, 1993.
- [23] Thomas Gutekunst, Bernhard Plattner: "Sharing Multimedia Applications Among Heterogeneous Workstations". *Proceedings, Second International Conference on Broadband Islands, pp. 173 - 191*. Athens, 1993.
- [24] J. Chris Lauwers, Keith A. Lantz: "Collaboration Awareness in Support of Collaboration Transparency: Requirements for the Next Generation of Shared Window Systems". *Proceedings, ACM CHI '90 Conference (Human Factors in Computing Systems), pp. 303 - 311*. Seattle, 1990.
- [25] J. Chris Lauwers, Thomas A. Joseph, Keith A. Lantz, Allyn L. Romanow: "Replicated Architectures for Shared Window Systems: A Critique". *Proceedings, ACM Conference on Office Information Systems, pp. 249 - 260*. Cambridge (MA), 1990.
- [26] Joel McCormack, Phil Karlton, Susan Angebrannt, Chris Kent: "x11perf". Software package in public domain. Copyright 1988, 1989 Digital Equipment Corporation. Available via anonymous ftp and internet information retrieval services.
- [27] Greg McFarlane: "Xmux—A system for computer supported collaborative work". *Proceedings, 1st Australian Multi-Media Communications, Applications & Technology Workshop, pp. 12 - 28*. Sydney, 1991.
- [28] John Menges: "The X Engine Library: A C++ Library for Constructing X Pseudo-servers". *Proceedings, 7th Annual X Technical Conference*. Boston, 1993.
- [29] Wladimir Minenko, Jean Schweitzer: "Transparentes Application-Sharing unter X Window—Synchrone Telekooperation für das Team von morgen". *Unix/mail, Vol. 12, No. 4, pp. 348 - 357*. Carl Hanser Verlag, 1994.
- [30] Adrian Nye: "X Protocol Reference Manual". O'Reilly & Associates, 1992.
- [31] T. Rodden, J.A. Mariani, G. Blair: "Supporting Cooperative Applications". *Computer Supported Cooperative Work (CSCW), Vol. 1, No. 1 - 2, pp. 41 - 67*. Kluwer Academic Publishers, 1992.
- [32] Mark Roseman, Saul Greenberg: "GroupKit—A Groupware Toolkit for Building Real-Time Conferencing Applications". *Proceedings, ACM 1992 Conference on Computer-Supported Cooperative Work (CSCW '92), pp. 43 - 50*. Toronto, 1992.
- [33] Robert W. Scheffler, Jim Gettys: "The X Window System". *ACM Transactions on Graphics, Vol. 5, No. 2, pp. 79 - 109*. ACM Press, 1986.
- [34] Kjeld Schmidt, Liam Bannon: "Taking CSCW Seriously". *Computer Supported Cooperative Work (CSCW), Vol. 1, No. 1 - 2, pp. 7 - 40*. Kluwer Academic Publishers, 1992.
- [35] Mark Stefik, Gregg Foster, Daniel G. Bobrow, Kenneth Kahn, Stan Lanning, Lucy Suchman: "Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings". *Communications of the ACM, Vol. 30, No. 1, pp. 32 - 47*. ACM Press, 1987.
- [36] Supoj Sutanthavibul: "xfig". Software package in public domain. Copyright 1985 Supoj Sutanthavibul, University of Texas at Austin, and 1991 Massachusetts Institute of Technology. Available via anonymous ftp and internet information retrieval services.



Thomas Gutekunst (S '92, ACM '93) received a diploma in Computer Science from ETH Zurich in 1991. In 1992, he joined the Computer Engineering and Networks Laboratory of ETH Zurich, where he is currently finalizing his Ph.D. thesis. His research focuses on the application of shared window systems in the context of collaborative multimedia for high-speed networks. His e-mail address is gutekunst@tik.ee.ethz.ch.



Daniel Bauer received a diploma in Computer Science from ETH Zurich in 1993. In 1993, he joined the Computer Engineering and Networks Laboratory, where he works towards his Ph.D. His interests are in new design approaches for communication protocols. In particular, he is interested in protocols for multicasting multimedia information. His e-mail address is bauer@tik.ee.ethz.ch.



Germano Caronni (AF '94) received a diploma in Computer Science from ETH Zurich in 1993. In 1993, he joined the Computer Engineering and Networks Laboratory, where he works towards his Ph.D. His research is concerned with security issues in communication systems, with a particular interest for dynamically configurable security architectures. His e-mail address is caronni@tik.ee.ethz.ch.



Hasan studied Electronics at the Politeknik ITB in Bandung, Indonesia from 1982 to 1985. From 1986, he participated in an exchange student programme which allowed him to enroll as a Computer Science student at ETH Zurich, Switzerland. In 1993, he received a diploma in Computer Science from ETH Zurich. Also in 1993, he joined the Computer Engineering and Networks Laboratory as a research assistant. In 1994, he left Switzerland and went back to Indonesia.



Bernhard Plattner (S '76, M '83, ACM '76), is a Professor of Computer Engineering at ETH Zurich, where he leads a communication systems research group. He received a diploma in Electrical Engineering from ETH in 1975 and a Ph.D. in Computer Science in 1983. His research currently focuses on applications of communication systems and higher-layer protocols. Specifically, he is involved in the development of multimedia applications for high-speed networks and is exploring new approaches to protocol engineering. He is also a member of the Internet Society and Vice President of TERENA, the European association of research and education networks. His e-mail address is plattner@tik.ee.ethz.ch.