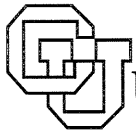
Software Support for a Virtual Planning Room

Gary J. Nutt
Joe Antell
Scott Brandt
Chris Gantz
Adam Griff
Jim Mankovich

CU-CS-800-95



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

Software Support for a Virtual Planning Room

Gary J. Nutt, Joe Antell, Scott Brandt, Chris Gantz,
Adam Griff, and Jim Mankovich

CU-CS-800-95 December 1995

University of Colorado at Boulder

Technical Report CU-CS-800-95
Department of Computer Science
Campus Box 430
University of Colorado
Boulder, Colorado 80309

Software Support for a Virtual Planning Room

Gary J. Nutt, Joe Antell, Scott Brandt, Chris Gantz,
Adam Griff, and Jim Mankovich

December 1995

Abstract

Computer supported cooperative work depends on conventional personal productivity tools and on the ability of the system to provide effective means for various members involved in the work to interact with one another. The interaction mechanism may be formalized through operations on shared information, through strict message-passing protocols, or by providing a means by which the human users can interact with one another through conventional conversation. Desktop virtual environment technology can be used to create group human-computer interface in which different users on different computers interact with common virtual artifacts, and with representations of one another using pointing devices, keyboards, and bitmapped screens. Virtual environments place tremendous demands on bandwidth for data movement, extensive graphics, data stream synchronization, and frequent interactions among object representations of the artifacts and virtual users. This paper describes our organization of an experimental virtual environment, called the Virtual Planning Room.

1 Introduction

Traditionally, computers have been used to assist organizations as generic information processing tools. Today there is a strong interest in using computers to assist groups of people to work on a common problem [8, 14, 16]. However, there is a diversity of opinion about how computers might best assist a group in its work, ranging from "situated work" [26], to automatic coordination systems [20]. There is an emerging camp that takes an intermediate position: the computer might be used to assist in computation and the coordination or work, though the team should not rely on the machine to do all the computation nor all the coordination for the work. This approach to computer-supported cooperative work (CSCW) provides facilities to support both structured and unstructured work conducted by a group of humans. In another paper, we argue that *model-based virtual environments* (MBVEs) are especially well-suited to CSCW work because they provide a framework in which the group can capture the structured parts of their work, yet still have a high bandwidth mechanism for formal and informal communication [24]. Like conventional electronic meeting rooms and virtual environments, MBVEs are compute- and I/O-intensive distributed systems; besides their inherent functionality requirements, reasonable system performance is a prerequisite to their success. This working paper describes our organization for the MBVE software, called the *Virtual Planning Room* (VPR); a subsequent paper will focus on the system software required to support it.

1.1 Rationale for the VPR

Group virtual environment applications depend on a robust software infrastructure. This infrastructure defines the means by which the collective applications read, write, and manage information within the environment defined by the infrastructure. Virtual environments must provide tools for managing information as conventional data, formatted documents, graphics, images, image streams, audio streams, etc. The collaboration infrastructure extends this support so it can be used in a distributed system with many human users.

Virtual environments extend the notion of electronic meeting room by creating a human-computer interface to simulate a virtual meeting room/space/environment in which human participants can logically immerse themselves. Interactions among the electronic surrogates are then conducted within the virtual space. In particular, artifacts in the virtual environment are then directly shared by the participants via interactions among the set of all artifacts and electronic surrogates in the environment.

The VPR establishes a multimedia electronic meeting room. It differs from conventional electronic meeting rooms in that human users need not be present at the same physical place. Instead, each user interacts with all other users via their own private workstation and an interconnection network. The collection of workstations, augmented with various network services provided by other nodes on the network, is the physical meeting room.

Since we are interested in supporting several different media for collaboration, each user workstation is assumed to be configured to provide substantial computational power, secondary storage, support 3-d color graphics, a microphone, moving image camera, and speakers. While the VPR is an instance of a virtual reality system, we do not assume the presence of an immersive head-mounted display for the workstation. Instead, we consider *desktop virtual environments* where images of the virtual environment are rendered on a 2-d screen using 3-d graphics, e.g, similar to the facilities provided on SGI Indigo workstations. In the future, we expect our work to apply to

visually immersive systems, though we do not assume sufficient graphics processing power in the user's workstation to support the various devices mentioned above and a head-mounted display.

1.2 Design Issues

Desktop VE systems are complex compute- and I/O-intensive application software. The complexity arises from the breadth of function incorporated into this software — it must use complex communication protocols, provide data abstraction tools for diverse media types, drive I/O-intensive displays and input devices — and the nontrivial computation to represent movement and interaction of objects in the virtualized space. Specific application software within the VE may add considerably more complexity to the software. From a practical standpoint, an open systems approach is mandatory in constructing these systems to enable VE programmers to pick and choose among various VR components that implement the details of the human-computer interaction environment, while they focus on the domain-specific part of the software. However, without cost-effective performance, the resulting software will have little effect on the contribution of the system to real problems.

The collaboration infrastructure must be built to fall within the bounds of various response time constraints; loosely speaking, one might characterize collaboration software as being subject to real time constraints, though the deadlines in collaboration systems are softer than those in traditional real time systems. If the system fails to satisfy these threshold performance constraints, it will not be a viable technology.

The intermediate- and low-level software and hardware must provide the most efficient platform possible for the VE, yet still provide a robust set of functions. Further, we recognize the value of open systems solutions in this environment, since there are many different approaches to supplying a collaboration infrastructure. Therefore at the low-level, we assume the existence of widely-embraced, open, object-oriented collaboration infrastructure that, in turn, relies on "standard" services of underlying object management, operating system, and network interfaces such as CORBA, UNIX, and ISI/BBN, respectively.

Today, the full computing resources of one or more high-end graphics workstations is required to support immersive VE. Desktop VE is less graphics-intensive than immersive VE, yet it still requires a high end graphics workstation, with extensive memory, cycle time, and graphics hardware. Such machines are almost universally configured with UNIX operating systems.

The UNIX operating system was designed as a timesharing system, thus its strategy for scheduling, blocking processes, data throughput, etc. is not particularly well-suited for VE middleware and applications. Jeffay [19] and others have posed alternative operating system strategies tuned to support multimedia-based domains. Independently, there is a trend in operating systems design to factor conventional operating systems services into a minimum nucleus of functions called a *microkernel* with full services being provided by *servers* that use the microkernel. Most such microkernel studies provide a UNIX server so that the microkernel and server implement a UNIX system call interface. For our VE, we propose to select a "standard" microkernel (e.g., Mach, Spring, Chorus, or Amoeba) and design specific servers tuned to support VE middleware and application demands.

There are several difficult software problems to be addressed in this study, including:

The Architecture for the VPR. There are many toolkits available for creating a virtual environment, though it is clear the choice of a sound software architecture for multiperson virtual environments is still the subject for investigation (e.g., see the proposer's information packet

for the Collaboration Technology area of the ARPA BAA 96-06 on High Performance Distributed Services Technology). What is a reasonable collaboration infrastructure to represent the VPR where one can add "applications" to run in the room?

Embedding Significant Applications in the VPR. Application-specific processing within a VPR introduces several new issues e.g., regarding the "spatial scope of the application" — if a VPR contains a pedestal on which a Petri net can be constructed, what are the properties of the Petri net components if they are moved away from the pedestal? Can they still be viewed as Petri net places and transitions? What, if any, editing operations are supported outside the immediate spatial area for the application?

Distributed Objects The VPR is a client-server computation, where clients support individual designers, and servers provide shared services. Objects must be sharable among clients, though they may be managed by servers. Our work suggests we need private objects in a client, shared objects in a client, shared objects in a server, and cached objects in a client (backed by a server). Further, cached objects must have a variable policy for coherence, depending how it is being used in the VPR.

Interactions Users at client workstations interact with objects in the virtual space. For example, a group of designers might be discussing and modifying a set of nodes and edges in a 3-d graph. How should interactions with objects be handled efficiently? What kind of support should the infrastructure provide to assist in interaction? Is it possible to distribute an object's interaction policy between clients and the object server?

1.3 Related Work

There are many experimental, government, and commercial systems, components, and products using virtual reality, virtual environments, multimedia, and desktop videoconferencing technology. The vast majority of these system and studies are single-user systems, though there is an increasing thrust toward ones focusing on collaboration in the virtual space as we do. Boman provides a recent survey article on virtual environments describing several systems with characteristics similar to the VPR described in this proposal [3]; the July, 1995 issue of IEEE Computer focusing on virtual environments [17]; the May, 1995 issue of IEEE Computer focusing on multimedia [18]; the AT&T Technical Journal on multimedia [1]; and the ACM Multimedia proceedings [10].

There are several specific studies and technologies that have influenced the design directions we have chosen. Next we briefly describe the most influential of these studies.

Defense Modeling and Simulation Office Activity. The DMSO sponsors several applied research projects, including various virtual environment projects to simulate defense exercises [4]. The E2DIS virtual environment is an object-oriented virtual environment addressing many of the same issues we describe in our proposal [5]. The NPSNET focuses more on network issues, also overlapping our proposed work [23].

Swedish Institute of Cognitive Science DIVE System. DIVE is a prototype distributed virtual reality environment to support conferencing [9]. DIVE focuses on issues such as how participants can determine one another's presence, how can a virtual person's immediate surroundings

be transmitted to the human user, and what is the importance of proximity to an artifact in virtual space.

AT&T Multimedia Research Platforms. Berkley and Ensor describe three research projects in AT&T Bell Labs relating to our VPR [2]. The most relevant system is the Rapport system with various MR (meeting room) extensions. Rapport is a basic desktop videoconferencing system supporting shared objects as in the VPR. The system explicitly recognizes the idea of objects in a Rapport meeting room are shared among participants, rather than having a private window onto shared information. In particular, the room can have a 2-d computer display shared by participants, just as we indicate the VPR can have shared X terminal displays in the virtual meeting environment. The MR extensions include a facility called "Visual Meeting Minutes." This VMM is early experimentation with marking and capturing snippets of electronic meetings. It appears as though the snippets are saved in a directory service (CORAL) rather than being embedded in a domain-specific model. The Archways 3D Visualization System is another extension from Rapport and MR to create a virtual meeting space with common objects. The Archways meeting room is similar to the VPR meeting room, though none of these tools are domain-specific as we propose in our work.

Bellcore Cruiser System Cruiser is a desktop audio-video conferencing prototype based on video telephony [11]. It is intended to allow people to communicate informally with a minimum of overhead. It explicitly provides facilities to allow participants to share computer applications (provided the applications have been constructed to be shared).

DEC Argo. Argo is another desktop videoconferencing system to support small groups [13]. Argo uses various off-the-shelf components (such as MBone and shared X servers) to provide a comprehensive videoconferencing system. We are influenced by their approach, but intend to embed more environment in the system than is done in Argo.

Xerox Etherphone. The Etherphone system is a multimedia system incorporating audio, video, and data in its environment using the Ethernet to integrate various communication media [27]. The Phoenix extension enhances the basic facilities to implement full audio-video conferencing. This work is important because of its contribution to system issues in collaborative multimedia systems.

Distributed Objects in VR. Funkhouser describes his work on techniques for efficiently managing interactions among artifacts in a virtual space [12]. His RING system maintains visualization regions (or domains) so interaction information can be propagated among artifacts where a visual interaction can effect behavior. A client-server architecture (similar is general characteristics to the one we propose though different in specific approaches and assumptions of types of interactions), is used to "cull" broadcast message traffic so interaction messages are propagated only to clients for which the update is meaningful. This work is a valuable starting point to our system architecture work.

Multimedia OS Support. Nakajima and Tezuka describe work being conducted at the Japan Advanced Institute of Science and Technology with researchers at Carnegie-Mellon University on

the use of RT-Mach to support multimedia applications [21]. Jeffay reports on his YARTS real time operating system kernel to support multimedia applications in [19]. The ACM Multimedia 94 conference includes a session with papers describing the synchronization problems related to combining audio and image streams [25].

Network Protocols. Ferrari and his associates in the Tenet Group have an ongoing research program focusing on high-speed computer networking [6]. The Tenet Group has participated in the BLANCA project, the Sequoia 2000 project, and the BAGnet project. This research has produced considerable new understanding about the use of networks to support multimedia applications, including those of the class of the VPR. We see the Tenet Group as one source of contemporary network protocols to be incorporated into the custom software system.

There are a number of other sources of network tools and protocols, e.g., see [1, 10, 15, 22].

2 The Virtual Planning Room

The VPR is a virtual environment providing various artifacts of interest to users of the environment, along with representations of the users themselves. We distinguish between the implementation of the representation of a human user from that of all other artifacts in the virtual space.

2.1 VPR Artifacts

Every artifact in the virtual environment — a wall, a pedestal, a pallette, a panel, a virtual terminal — is represented in the logical VPR as an object. The object provides components to represent the artifact's appearance and and its behavior. Some parts of the system are only concerned with the appearance (e.g, the workstation that renders the object), while other parts of the system may only be concerned with behavior (e.g, the aspect of the system responsible for determining situations where objects are interacting with one another). The VPR defines an environment in which these collective artifacts (represented by objects) logically reside, and in which a user can navigate, browse, and interact with artifacts he/she encounters in the environment. The implementation represents this scenario by manipulating objects in workstation and server machines.

2.2 The Virtual User

A human participant is represented in the environment as a a *virtual user* (also called an *occupant object* or simply "occupant"). When a human participant wishes to logically enter a specific virtual environment, then an instance of the virtual environment (called a *session* or *world instance*) is created if one does not exist, then an occupant object is created and placed in the session. The user's occupant is used to represent the user's interaction with the environment. When the occupant is created, it is placed at a specific location in the environment with a specific orientation (part of the occupant's state). The occupant has an active subobject component, called a *hand*, and a passive component, called an *eye*. The hand is the human user's surrogate tool for interacting with other artifacts in the VPR. The eye establishes a location and orientation from which the human user views the virtual environment image. The eye image is a 2-d perspective view of the part of the environment visible from the eye's logical location and orientation.

There is a default relationship between the location/orientation of the eye and the location/orientation of the hand. Changing the eye location/orientation explicitly changes the hand location/orientation. However, the hand may be moved without changing the eye location and orientation, e.g., to grasp an object in the eye's view.

The hand also contains a "ray" extension. This simplifies artifact manipulation by allowing the hand to "shoot a ray" along a vector determined by the hand's location and orientation. For example, a user's hand can push a button on a panel by orienting the hand to point at the button then by shooting the ray.

The image perceived by the eye is arbitrarily realistic, depending on the workstation system and eye implementation. For example, our single screen-based eye provides a perspective image to represent a 3-d view of the part of the virtual environment perceivable to the eye from its location and orientation. The nature of the graphics employed in the eye implementation is chosen on the basis of performance-quality tradeoffs. For example a monoscopic eye implemented in an SGI GL environment can represent color images with lighting, shading, etc., though artifacts may be represented as simplified icons. An Intel-based processor in conjunction with Mesa and OpenGL renders the image in an X terminal environment, thus the images are even less sophisticated due to the amount of time required to create and render them in this computing environment.

As mentioned above, the human user moves the (hand, eye) pair by moving the eye within the virtual environment. Physically, the workstation provides a logical *eye tracker* device to represent the location and orientation of the eye. The eye tracker may be implemented using a keyboard, mouse, or other specialized device (e.g., a head tracker on an immersive helmet). Similarly, the location and orientation of the hand is controlled by a logical *glove*. When the human user moves the glove device, the hand object moves to represent the location and orientation of the hand in the virtual environment. The glove may also be implemented with an arbitrarily simple physical device.

Summarizing, the virtual user is represented by a pair of objects, (hand, eye), associated with physical input devices. Both the hand and eye objects have a location and orientation in virtual 3-space, represented by a 6-tuple (x, y, z coordinates, and coordinates for yaw, pitch, and roll angles). The human user manipulates the virtual hand and eye by manipulating physical devices which map to the hand and eye via the eye tracker and glove devices. The eye object determines its location/orientation, the set of artifacts visible in the virtual space, then renders an image on the screen(s). When the occupant location/orientation or environment change, the image is recomputed and redisplayed. The hand object tracks the eye tracker and the glove. The glove can be used to shoot a ray according to the hand's current location/orientation, or to grasp/release an object within the vicinity of its location.

2.3 Application-Specific Behavior

An application program in a VPR is implicitly different from traditional notions of application program. The occupant object is a composite object able to navigate through an environment of artifacts located in a virtual 3-space, each represented by other objects having their own appearance and behavior. It is tempting to require the objects in the environment to encapsulate application-specific behavior, though that approach is problematic since the occupant would then be required to interact with the application behavior via a robust software interface; we do not currently have the experience to design such an interface. Therefore, our initial approach is to embed part of the

application in the occupant objects and the remainder in (non-occupant) artifact objects. This is not defensible in the long term, since it is not scalable (though we acknowledge that it is similar to the case with humans, i.e., a human must acquire and retain knowledge about how to use tools before using them). Our longer term goal is to define a reasonable interface between the occupant objects and all other objects intuitively corresponding to a traditional human-artifact interface. For now, the VPR system requires the programmer to extend the behavior of a virtual user, e.g., by providing basic behavior that can be inherited by new classes.

For example, if the VPR were to contain a graph model editor, part of the functionality of the editor would be implemented in the objects representing the modeling artifacts and the remainder in the occupant objects. This causes the occupant objects to become arbitrarily complex — effectively carrying the union of all behaviors for all applications the occupant uses. This allows us to build nontrivial applications in the VPR without having to know how to design an application-independent interface for the occupant class.[1]

2.4 Sessions

There are two important aspects regarding construction of a VPR session. First, artifacts appearing in an environment must be designed. Second, a VPR world must be configured in which occupants navigate and interact with other occupants and with artifacts. The VPR world specification defines the virtual space and all artifacts in it using a script. The behavior of the VPR — its applications — is defined by the collective behaviors of the objects implementing the artifacts in the VPR.

A session is an instantiation of a VPR world through the creation of a virtual space containing various artifacts implemented using objects. Thus, each session is a logical meeting among a group of human users at workstations in a specific virtual environment. Notice, different sets of users could be meeting in different instances of a world; occupants in different sessions would not interact with each other nor even with any objects derived from common artifacts.

If two or more users `join` the same session, they will be placed in the same world instance. They may also `join` different sessions derived from the same world definition (artifact class implementations and VPR definitions), meaning that they are in different virtual spaces having distinct copies of the same artifacts.

We have adopted a simplified model for coherence of objects in simultaneous sessions similar to the UNIX file system semantics. In UNIX, it is always possible for multiple users to open a file for writing at one time; however, only the last user to close the file will have their version of the file saved, since a close simply overwrites whatever was in the file previously. In the VPR it is possible to `save` a session's state by extracting the world representation from that session's state and writing it back to the world definition. If two or more sessions are open using the same world definition, then the last session to perform a `save` operation will provide the last update of the world specification. We also support an option to extract a world representation and to save it under a new world name (i.e., we also provide a `save as` function). This permits human users to define a new world representation from a particular session with an old world. Similarly, a session can be ended with no save operation.

[1] We also note it might not ever be possible to build occupant classes that do not possess knowledge of applications, since this would be analogous to having humans using systems where they never required training about the tools prior to using them; obviously not possible in the real world.

Our design presumes a client-server architecture with the logically shared aspects of the environment managed by servers and individual human-computer interactions occurring at client workstations. The client supports and controls its user's perception of the VPR, but the server controls the session and shared artifact semantics, e.g. interactions. Different users may see different visualizations of the logical VPR depending on the client implementation they are using.

A virtual room/space is defined by first defining a collection of classes representing different kinds of artifacts used by the corresponding session. A session is instantiated by executing a script to create the desired set of objects representing the artifacts in the virtual space.

Objects can be introduced into a session using a *replicator* facility. The replicator deals only with the object's appearance; it is intended to be used to obtain a VRML description of some artifact, then to create an object with a default location, orientation, and scale. The replicated object has no default behavior, so that must be added to the replicated object using other facilities. At this time we have not designed any tools for adding behavior to a replicated object.

2.5 VPR Snippet Capture and Management

Information describing a meeting in the VPR is represented as a set of objects, operated on by various clients. This information can be captured to archive any (part of a) session in a VPR. We do not currently address the nature of the storage hierarchy required to support a production VPR, but rather, assume a flat file space to store meetings and meeting snippets.

Part of the capture technology must also include an annotation facility to allow users to mark snippets and to attach a set of reference keywords. It should also be possible to mark and annotate after a session has been completed. Our initial approach to this function will be patterned after conventional methods for marking scanned images and videotapes.

The modeling system will then be modified to allow users to attach snippet references to various artifacts in the VPR. A browser can then be used to inspect the artifacts, including following pointers to meeting snippets.

3 VPR Software Organization

The VPR system is built on a client-server architecture with sessions, artifacts, and snippet management logically implemented on the server, and human-computer interactions occurring at individual client workstations (see Figure 1). The basic principal for selecting whether a function is implemented in client or a server relates to whether or not the result of the function is shared across users or is specific to the user at the workstation. For example, the client supports and controls its user's interaction with the occupant, i.e., its hand-eye interaction with the VPR. Similarly, a server manages the creation and admission to any particular world.

A world is a set of objects representing all the artifacts in a virtual environment. For example, if the environment is a single room with 4 walls, then the world contains a set of objects to represent the walls; if the room has floors, doors, windows, etc., then objects must be defined to represent each of the artifacts. Every object has properties, and potentially some behavior. For example a door object has visual properties such as color, translucence, location, orientation, etc. It may also have behavior, e.g., such as opening on demand or closing itself after an arbitrary amount of time.

An important consideration of VPR objects is their *appearance* as well as their *behavior*. We use the VRML scene markup language to define artifact appearance within a more general object

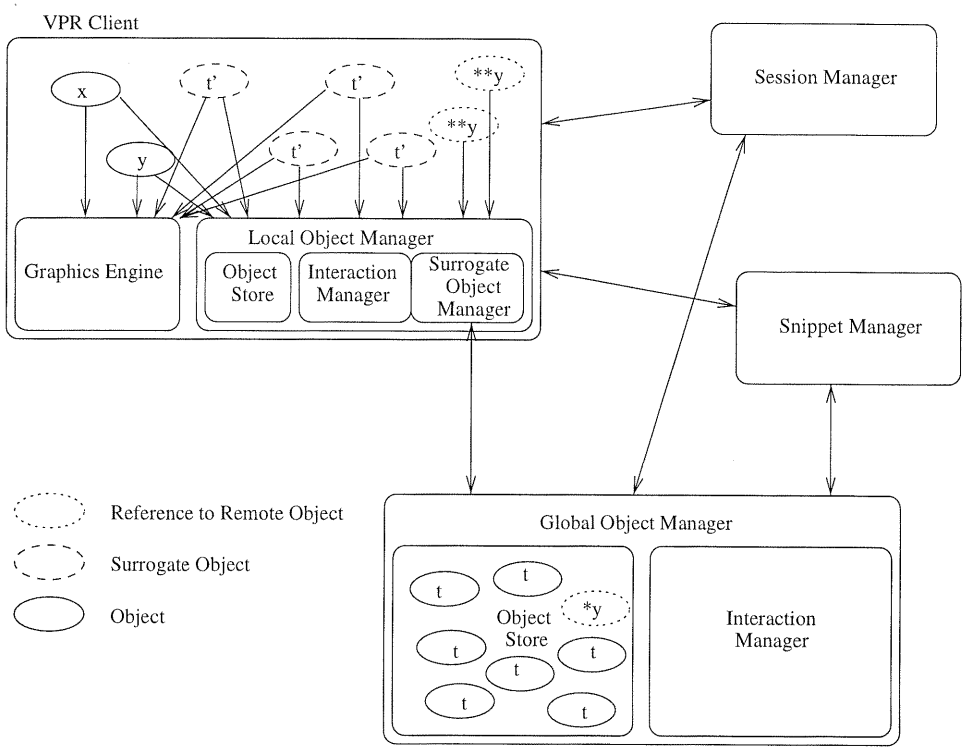


Figure 1: The Clients/Servers Organization

framework so we can also represent the artifact's behavior. If an object has only appearance, but no particular behavior, it can be introduced into the world by creating a VPR object with a minimum of VPR information (e.g, location, orientation, mass, and volume) and the VRML description of its appearance. Thus, objects with arbitrarily sophisticated VRML descriptions can be obtained from the computing environment (e.g., from the Internet), embedded in a minimal VPR object, then be rendered in a world. The location, orientation, mass, and volume are minimal properties used to determine how other objects will perceive and interact with the object.

3.1 An Execution Scenario

A human user decides to use the VPR by first knowing the name of a session they wish to join. The user begins to use the VPR by running the client software on the local workstation. The client facilities include a graphics engine to render the eye's view, a set of objects to implement the world as perceived by the occupant, and a local object manager (see Figure 1). The client computing environment will also include various other facilities to implement device drivers, interface to the network, etc. When the client is started, it is not associated with any world, thus no world is visible to the user. Instead, a dialog ensues in which the user specifies the identity of a session he or she wishes to join. The client passes the request to the session manager.

The session manager is responsible for mapping a session request to a world representing the session. That is, in principle, the session manager keeps a copy of the world in a location where it can easily be retrieved, e.g, in a network file system. The session manager creates an instance of the global object manager at an arbitrary network location (in our first implementation, that location

is the first client's machine), then informs the global object manager of the client request to join a session. The session manager then returns information to the client informing it of the network location of the global object manager for the world. At this point, there is one client and the global object manager in operation. The local object manager in the client is prepared to interact with the global object manager using a network interface.

The global object manager downloads information to allow the client to reference every object in the world associated with the session. Some objects may be cached into the local object manager's storage, while others may be allocated at other places on the network, e.g., in the global object manager's storage. The client creates the occupant object in the local object manager's storage, then passes a handle for it to the global object manager. Then computes the appearance of the world from the location and orientation of the occupant eye. The client is then ready to respond to the occupant's directions (i.e., to the human user's commands to the VPR through the occupant via the head tracker and glove). We also use the workstation's keyboard and mouse as input devices to the occupant or the local VPR software.

Suppose a second client joins the session. The session manager responds to the client by telling it the network location of the global object manager. The new client establishes contact with the global object manager, obtains information about the session, creates its occupant, reproduces its view of the world, then waits for something to do.

Each client is driven by its own user's commands and by the actions of an occupant on another client. For example, if a remote occupant moves an artifact in the world, it must move in all client worlds that "care about" the object. Hence, object movement potentially effects many different world copies, thus we rely on the global object manager to provide a protocol/implementation to manage movement. The interaction manager aspect of the global object manager is a logically centralized authority for determining generalized interaction between two distinct objects. In particular, if an occupant moves an artifact (including itself), it must have its movement authorized by the interaction manager. Note that the interaction manager can be distributed according to what logical locality, e.g., if an object is visible to an occupant in a world, then the client machine must be concerned with interactions — by its own occupant and by others. Our current implementation uses a simple, centralized protocol by which interaction management is implemented as a centralized service on a single global object manager for the world.

The snippet manager is a storage server. A human user can mark a spatial and temporal segment of a world session, then store the collective object states and their transactions in the snippet manager. Each snippet has a unique identification that can be referenced from any world instance. In our current design, the snippet manager provides no additional facilities.

4 Sessions

A *session manager* launches a session by starting a new global object manager, and suspends dormant sessions at the request of the global object manager. The session manager also saves closed sessions on its own long term storage. A client becomes part of a session by executing a conceptual code sequence such as:

```
session = sessionMgr.lookup(char *sessionName);
session.join(Occupant self);
```

This function call on a client function causes a datagram to be sent to the session manager to join the identified session.

If a session has never before been invoked, the global object manager is started, creating the VPR objects that exist in the world associated with the session. This potentially requires the session manager to retrieve VRML descriptions from arbitrary servers, and to create VPR objects with their embedded VRML field.

If a session has been created, but is currently dormant (i.e., there are no users currently using the session), then the join request causes the session manager to look up the previously-defined world on the network.

Once the session's world description has been identified, the session manager reads the client's network address to determine where the global object manager will be created. It then establishes a TCP connection to the client machine and creates a global object manager on that machine. The session manager then responds to the join request by presenting the client with the network connection to the global object manager (on its own machine). At this point, the session manager has completed the join.

The global object manager handles objects created for a session's use. When the session becomes dormant, the system maintains a description of the world. The global object manager can save a session by serializing the object descriptions and passing the information to the session manager. It stores the description on its network-accessible storage. The session manager can also be used to save a session state at any time. This is accomplished by having the global object manager serialize its objects and writing them to the session manager (changing the world description on the session manager).

The session manager uses only simple semantics for managing world state. For example, if a client 1 starts a session, executes for a while (changing the object state), then client 2 starts a new session from the same world, it will use the same world state as client 1 used when it started. Changes in state caused by the two different sessions are not managed; the last session to checkpoint its state will update the world description.

5 The Client Computing Environment

When a client joins a session, it will be provided with the network address of a global object manager. It will also be responsible for creating a *perspective* in the world by creating an occupant with an explicit location and orientation for the occupant's hand and eye. This information is presented to the global object manager with a request for appropriate objects. The global object manager uses its policy to decide which objects will be needed by the client, then provides mechanisms to reference each such object. Its policy determines if the object is copied to the client or if a global pointer is provided by which the client references the object indirectly. The client then constructs the eye's view from the object list.

5.1 Creating Atomic Objects

Objects are loaded into a client's environment from the global object manager as composite objects with a VRML component and basic positioning information. The client unpacks the composite object, then uses the VRML in conjunction with atomic object classes to create a set of lower level objects that make up the composite. The client parses the VRML script, generating a VRML parse

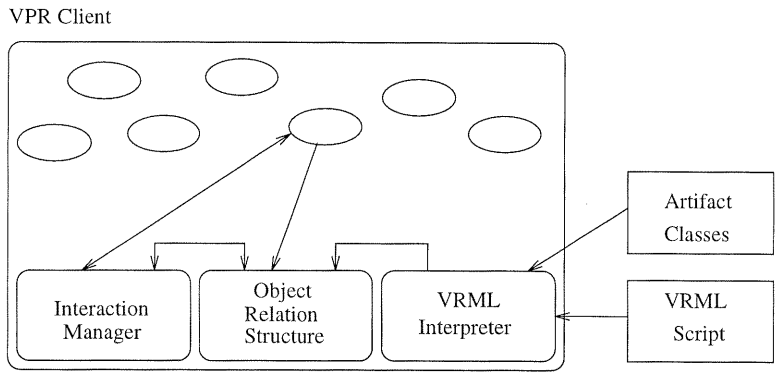


Figure 2: VRML and the Application State

tree (see Figure 2). A VPR VRML interpreter function traverses the parse tree, generating objects using the appearance derived from the parse tree and behavior derived from artifact classes. If there is no artifact class for a derived object, then the object will have only appearance attributes. Thus, the client transforms the composite object VRML parse tree into an internal *atomic object relation tree*, describing the objects and relationships that exist among them, e.g., their display order and interaction dependencies.

Once the internal data structure is established, any object that behaves like an editor on shared objects must update the object relation tree. However, before it can do so, it must have the operation approved by the interaction manager. In particular, if the system can be designed so that only objects of type `Occupant` change state, the control over shared object updates can be much better controlled than if arbitrary objects change other objects.

6 Object Management

As mentioned in Section 2, every object has a behavior, and it potentially has an appearance. We assume every visible object has a location/orientation, represented as a 6-tuple defining its location and orientation in the virtual environment coordinate space. We distinguish between objects representing a human's presence in the virtual environment and all other objects. First we discuss generic objects, object appearance, object behavior, object management, then objects representing humans.

6.1 Generic Objects

Generic objects can further categorized, based on their ability to perform operations in the VPR and on the kinds of operations to be performed on them (see Figure 3).

The characterizations in the figure are defined as follows:

Active vs Passive. An active object is capable of autonomous actions, including interacting or establishing a dialogue with other objects. The intuitive aspect we wish to capture with this distinction is that active objects must explicitly manage interactions, but passive objects are handled by active objects. For example, a virtual person is an active object. A passive object

Type	Active or Passive	Have mass?	Have volume? (visible?)	Aural presence
1	Passive	no	no	yes
2	Passive	no	yes	yes
3	Passive	yes	yes	yes
4	Active	no	no	no
5	Active	no	no	yes
6	Active	no	yes	yes
7	Active	yes	yes	yes

Figure 3: Object Categories

responds to interactions initiated by an active object, e.g., an X terminal display is a passive object.

Presence. How can the object be perceived by other objects? An object's presence can be detected if it makes sound, if it is visible, or if it occupies space. These characterizations are separated:

- **Mass.** If the object has mass, it is impenetrable by other objects having mass. An object with no mass can be located at the same point in the virtual space as an object with mass. If two objects have mass, they cannot be located at a common point in the space. If an object has mass, its presence can be detected by attempting to move into the space the object occupies.
- **Volume.** If the object has volume, it has a visual image perceivable by other objects. I.e., its presence can be detected by observing the space it occupies. An object can have volume but no mass, but if it has mass it must have volume.
- **Produces Sound.** If the object produces sound, it can be heard by other objects. Its presence can be detected by listening for the object.[2] An object may produce sound but have no mass nor volume. If an object has volume, it is assumed to be capable of producing sound.

Any object may or may not be able to sense (e.g., see or hear) other objects. Conversely, an object may have properties allowing its presence to be perceived, e.g., by having volume or by producing sound. This leads to the categorization of object types shown in Figure 3.

Type 1. A Type 1 object is a passive object that can be heard, but will not emit sound unless directed to do so, e.g., an audio recorder.

Type 2. A Type 2 object can produce sound, and can be seen. As a first cut definition, a Type 2 object can have its location in the virtual space changed by some active object. Since a Type

[2] We have not included an "ear" for the virtual person, but expect to map its location and orientation directly to the eye.

2 object has no mass, other objects can occupy the same part of the virtual space as a Type 1 or 2 object. A voice channel to other VPR Occupants is an example of a Type 2 object.

Type 3. A Type 3 object can be seen, heard, and occupies a measurable area in the virtual space. If a Type 3 object is in an area, no other object with mass can have any part of its volume in the same space. Type 3 objects are the most common kind of passive object, including walls, pedestals, X terminal displays.

Type 4. A Type 4 object is an active object, but it cannot be perceived by other active objects. The first version of a virtual person is a Type 4 object. It can navigate in a VPR observing and possibly interacting with other objects, though it cannot itself be perceived by other active objects.

Type 5. A Type 5 object differs from a Type 4 object by being able to speak — meaning its presence can be perceived by the sound it makes (if it makes any sound). This type may not result in any particularly useful artifacts.

Type 6. A Type 6 object differs from a Type 5 object in that it can be seen, but is ethereal (has no mass). Occupants could also be Type 6 objects.

Type 7. A Type 7 object is an active object that has volume and mass, so if it occupies an area in the virtual space, no other object with mass can occupy the same space.

6.2 Object Appearance

An object's appearance is defined by a VRML script. The appearance is a member field for an object, and can be arbitrarily complex. The global object manager does not interpret the appearance, leaving that aspect of the object to the client.

When the client intends to render an object in the virtual space, it uses the VRML definition to do that. In addition to the aspects of the VRML script defining appearance, there are certain other fields that define the placement of the object in the space, e.g., translation, rotation, location, and orientation. Whenever an object is transformed, the local interaction manager aspect of the object manager is involved in approving the interaction. This may, in turn, require the attention of the global interaction manager part of the global object manager. Thus, there is one interface between the object manipulation module and the local interaction manager and another between the local and global interaction managers.

The VPR system relies on the existence of an OpenGL interface (e.g., as implemented in the Mesa software package). Just as editor objects must take special precautions to manage sharing by using the interaction manager, the interface between the graphics library and visible objects is complex. It must allow the object to interact with the GL engine to define light, material, reflection, absorption, etc. so that the object will be rendered with the full capability of the graphics library.

6.3 Object Behavior

An object's behavior is determined by various methods associated with the object. The global object manager creates an object with its specific behavior and appearance from the world description kept on the session manager. When the object is made known to any client, it is instantiated in the client and/or the server (depending on object semantics used by that object).

Object Sharing. From the programmer's point of view, objects representing artifacts in the shared environment are shared objects. Since occupants are implemented on different machines, the majority of objects in the environment are shared across machine boundaries.

Objects have *rules* to specify various facets of their behavior, with respect to the way they are shared. Suppose objects G and H are implemented on different workstations, and G requests H to perform an operation.

Mutability. Object H may be mutable or immutable with respect to the request from object G. For example, G_1 may be able to write object H but G_2 may not have such permission, meaning H is mutable to G_1, but immutable to G_2.

Static vs Dynamic. Objects may be static or dynamic. A static object is immutable to all objects. A dynamic object is mutable to at least one object in the environment. If an object is static, it can be cached into different address spaces with no concern about updating the copies when any one of them changes.

Consistency in Dynamic Objects. If the object is mutable in workstation α but immutable in workstation β, the copy of the object in β must be made to be consistent with the changes the object in α. If an object is mutable in α and β, there must be a policy regarding when changes can be made, how they will be synchronized, and how the changes will be propagated among the copies.

- **Coherence Granularity.** This rule addresses the policy for propagating update information from dynamic objects. It specifies the rate at which objects are kept coherent (low to high). It may also define the scheme for providing coherence information, e.g. asynchronous or synchronous.
- **Update Type.** When a dynamic object changes, it may be the responsibility of the object to "push" all changes to other interested objects, or interested objects may be required to "pull" update information from objects of interest.

6.4 Object Management

Each user operates in a local computing environment, conceptually implemented in a client workstation, interacting with the global computing environment implemented in servers. A session instance may have *private* objects local to the session instance, such the variable x in the client 1 program in Figure 4. *Distributed* objects are updated by this session instance (client), but perceived by other session instances in the same session; the variable y in client 1, *y in the server, and **y in client 2 is an example of a distributed object. A global *shared* object can be read and written by any session instance in the session. The variables *z and t' are different implementations of shared objects as perceived from a client environment: in the case of *z, the client has a handle to the object located on the server. In the case of t', the client has a copy of the object t located on the server.

Private objects are managed by the local computing environment, and distributed and shared objects are managed by a combination of network services and the local computing environment. Object management refers to specialized operations on objects to create, destroy, display, and change their state. Each of the local and global computing environments provide an *object manager*

```
client_1 {
  extern *z;

  type x; // Local to the client
  type y; // Locally stored, but visible to other clients
  type *z; // Pointer to shared variable on the server
  type t_prime; // Cached copy of server variable
}

server {
  extern *y;

  type *y; // Pointer to a variable on client_1
  type z; // Shared variable on the server
  type t; // Variable with cached copies on clients
}

client_2 {
  extern **y, *z;

  type **y; // Pointer to pointer on the server
  type *z; // Pointer to shared variable on the server
  type t_prime; // Cached copy of server variable
}
```

Figure 4: Object Storage Typed

for this type of management. To emphasize the requirements of the object manager, we refer to its parts as the *object store* and the *interaction manager* (see Figure 1), though we do not require they be implemented this way. The local object manager creates, stores, destroys, and updates the state of local objects.

Shared objects such as z and t in Figure 4 may be copied to the local environment as a surrogate shared object, t', or have a handle, *z, placed in the local computing environment address space used to request the server operate on the object. Local and remote objects use the same object service interface when they execute in a client. The local object manager passes shared object operations to a surrogate object manager and it cooperates with the true object manager located at a server.

The object rules must be accommodated to satisfy the goal object semantics. For example, the object manager must be able to handle static, dynamic, mutable, and immutable, objects The object properties result in read, write, push, and pull object behavior.

Interaction Management. Since there may be many objects and occupants in a session, the interaction among artifacts is managed by an *interaction manager*. This interaction manager arbitrates the colocation of objects with mass in the virtual space. For example, A.Hand might

implement the interaction manager function by having A.Hand check the virtual space where it is about to move to see if another object with mass already occupies the space; if not, A.Hand moves into the space.

The interaction manager has a client part and a server part. If the effect of actions by active objects can be managed by the local interaction manager, then it will do so. If the interaction references objects managed by the server, then the client part of the interaction manager uses a private protocol to have the server part of the interaction manager control the interaction. The level of interaction is determined from the object rules described above.

6.5 Occupant Objects

An occupant is an object representing a user in a virtual space. Conceptually, an occupant is the aggregate of an object of type `Eye` and another object of type `Hand`. The Version 1 VPR represents the occupant as a Type 6 object (an active object with volume and sound, but no mass). Since occupants are Type 6 objects, other occupants can see them. However, since they have no mass they can move any place in the virtual space without checking to see if the space is already occupied by another object.

An `Occupant` can be either a `Browser` or an `Editor`:

The `Browser` Base Class. A Type 6 browser object navigates among objects in a session according to user directives, and displays a view of the environment according to the viewing perspective of the occupant's `Eye` instance. The occupant cannot cause the state of any object other than its own `Hand-Eye` sub objects to change. So we have:

```
class Browser : Type6 {
// Moves Hand and Eye objects
    Hand    *hand;
    Eye     *eye;
};
```

The hand ray is an specialized exception to the lack of interaction by a browser, since it allows the hand to push buttons.

The `Editor` Base Class. Any type 6 object, including `Occupants`, can be an `Editor` object. An `Editor` object can create new artifacts, and request state changes to other objects. The `Editor` object is required to coordinate and state change requests prior to directing such requests at other objects in the VPR.

```
class Editor : Browser {
// Must coordinate shared state changes
};
```

A simple example of an editor object is an `Occupant` object A attempting to move a passive object, P, from one place in the virtual space to another. This action requires A.Hand to be placed within an arbitrary distance of P in the virtual space.[3], that A.Hand grasp P, that A.Hand change

[3]This arbirary distance is called the passive object's *aura*

its location, then A.Hand release P. In particular, P cannot be "thrown" or "hit" by A.Hand to cause independent motion by P.

Let us assume P has mass. When A moves P, it is A's responsibility to determine that the part of the virtual space through which A and P move does not contain another object with mass. For example, A and P cannot pass through a wall or some other passive object with mass.

7 Snippet Management

A snippet is a segment of a VPR meeting. It is an archived record of all the object interactions that occurred in the designated part of the meeting. We distinguish between *local snippets* and *global snippets* as follows: a local snippet is a record of the objects and their actions over a time period from the perspective of a particular client machine (and hence of a particular occupant object). When a user at client workstation w_i decides to capture a snippet, s/he *marks* the snippet at the human-computer interface for the workstation. For example, a human user may mark the snippet by turning snippet recording on at one moment in time, then by turning it off at a later moment of time. When snippet capture is enabled, the client workstation takes a snapshot of all objects accessible in the workstation (though not necessarily in the world), then begins to record all interactions among objects. This continues until the human user terminates the mark operation.

A global snippet is also defined by a mark operation, though the scope of the snippet capture is over the entire world rather than over the scope used by the workstation. If a global snippet has been captured, then in principle it could be replayed as a new session in which the occupant changes its scope in the replayed segment. This obviously opens several new questions about acceptable interactions with objects in global snippets.

The snippet manager is responsible for capturing and playing back VPR snippets at the request of a client. At this time, we have not created any ability for a human user to change a snippet. A local snippet may be captured by the client, but global snippets are captured by the snippet manager. If a local snippet is captured by the client, then it must be registered with the snippet manager before it can be used.

When a snippet is replayed, the human user may halt the snippet and browse the corresponding state. In a global snippet, this means the occupant can move around the space, inspecting various artifacts, perspectives, etc.

We have many open issues with the snippet manager design.

Capture. Local capture uses the local interaction manager to identify object changes, thus it can store each significant interaction in the session. Global capture requires the snippet manager to establish a snooping protocol with the world's object interaction manager. How can this be done efficiently?

Compression. Image streams dominate the amount of space used to save a snippet. Is it sufficient to use MPEG to compress video streams, or is additional innovation necessary? How can other objects and streams be associated with the compressed MPEG representation?

Interaction with Snippets. Noninteractive snippet playback is the obvious initial step in snippet management. Once a snippet has been captured and stored, it can be played back as a videotape of the meeting segment. Once we begin to support interaction, a new set of issues arise, e.g., suppose an interaction in the snippet replay changes the state of an object

so that it is inconsistent with the snippet; should this be allowed? How can it be correlated with the remainder of the snippet (cf. shared memory multiprocessor trace extrapolation [7].

Global Replay. Should global replay create a new, nested world within a particular world?

8 Status and Conclusion

The VPR is an experimental software system to implement multi user virtual environments. It provides an exciting new form of teleconferencing in which artifacts that appear in the virtual environment are models of artifacts in a real world, or are completely virtual (i.e., there is no physical counterpart). Virtual environments enable users to conceptualize appearance of behavior of different things without actually building them. Since these artifacts exist only in software, they are not subject to the normal laws of physics that apply to real objects. The upside on the investment is the increased value of a system employing VR. The meetings can incorporate artifacts that simply do not exist in the real world. We see the VPR as a fundamental enabling tool to support collaboration applications.

Meeting snippets can be marked and saved for future reference. The general approach of combining an informal communication medium with a formal model can move the formal model from the role of official archive to active design tool. While snippets have the obvious problem of being "too complete" of a record of a meeting, their use for capturing design rationale or other key points in a debate are obvious. However, despite the clear need for careful management of its use, we see this facility as being a fundamental building block for studying asynchronous meetings.

The multi user aspect of the system relies on difficult new technology: it makes extensive use of high bandwidth network protocols for supporting interactions across the client machines. It also requires sophisticated approaches for supporting effective means for occupants to interact with objects. This working paper describes our approach to designing a multi person virtual environment called the virtual planning room. We view the application domain as being of increasing interest to collaboration software designers, and as providing an interesting new set of requirements for systems software.

References

[1] Special Issue of AT&T Technical Journal on Multimedia, September/October 1995. Nikil Jayant, Technical Reviewing Editor.

[2] David A. Berkley and J. Robert Ensor. Multimedia research platforms. *AT&T Technical Journal*, 74(5):34–45, September/October 1995.

[3] Duane K. Boman. International survey: Virtual environment research. *IEEE Computer*, 28(6):57–65, June 1995.

[4] Defense modeling and simulation office web page. WWW Web page, http://www.dmso.mil/, 1995.

[5] Environmental effects for distributed interactive simulation. WWW Web page, http://www.dmso.mil/dmso/dmsoprojects/, 1995.

[6] Dominico Ferrari, et al. The tenet group recent and current research, 1995. Available http://tenet.berkeley.edu.

[7] Zulah K. F. Eckert and Gary J. Nutt. Tracing nondeterministic programs on shared memory multiprocessors. Technical report, Department of Computer Science, University of Colorado, January 1996. submitted for publication.

[8] C. Ellis, S. J. Gibbs, and G. L. Rein. Groupware: Some issues and experiences. *Communications of the ACM*, 34(1):38–58, January 1991.

[9] Lennart E. Fahlen, Charles Grant Brown, Olov Stahl, and Christer Carlsson. A space based model for user interaction in shared synthetic environments. In *Proceedings of Interchi '93*, pages 43–48, April 1993.

[10] Domenica Ferrari, editor. *Proceedings of the Second ACM International Conference on Multimedia*. ACM, 1994.

[11] Robert S. Fish, Robert E. Kraut, Robert W. Root, and Ronald E. Rice. Video as a technology for informal communication. *Communications of the ACM*, 36(1):48–61, January 1993.

[12] Thomas A. Funkhouser. RING: A client-server system for multi-user virtual environments. In *1995 Symposium on Interactive 3D Graphics*, pages 85–92. ACM, 1995.

[13] Jania Gajewska, Jay Kistler, Mark S. Manasse, and David D. Redell. Argo: A system for distributed collaboration. In *Proceedings of the Second ACM International Conference on Multimedia*, pages 433–440, 1994.

[14] Jonathan Grudin. Groupware and social dynamics: Eight challenges for developers. *Communications of the ACM*, 37(1):93–105, January 1994.

[15] Proceedings of the Conference on High Speed Networking and Multimedia Computing, 1994.

[16] Special Issue of IEEE Computer on Computer-Supported Cooperative Work, May 1994. James D. Palmer and N. Ann Fields, Guest Editor.

[17] Special Issue of IEEE Computer on Virtual Environments, July 1995. David R. Pratt, Michael Zyda, and Kristen Kelleher.

[18] Special Issue of IEEE Computer on Multimedia Systems and Applications, May 1995. Arturo A. Rodriguez and Lawrence A. Rowe, Guest Editors.

[19] Kevin Jeffay. On kernel suport for real-time multimedia applications. In *Proceedings of the Third IEEE Workshop on Operating Systems*, pages 39–46, 1992.

[20] Thomas W. Malone and Kevin Crowston. The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1):87–119, March 1994.

[21] Tatsuo Nakajima and Hiroshi Tezuka. A continuous media application supporting dynamic QOS control on real-time mach. In *Proceedings of the Second ACM International Conference on Multimedia*, pages 289–297. ACM, 1994.

[22] Proceedings of the Workshop on Network and Operating System Support for Digial Audio and Video, 1994.

[23] The NPSNET research group home page. WWW Web page, http://www-npsnet.cs.nps.navy.mil/npsnet/index.html, 1995.

[24] Gary J. Nutt. Model-based virtual environments for collaboration. Technical Report CU-CS-799-95, Department of Computer Science, University of Colorado, Boulder, December 1995.

[25] Jonathan Rosenberg. Session 3a: Synchronization. In *Proceedings of the Second ACM International Conference on Multimedia*, pages 133–156, 1994.

[26] Lucy A. Suchman. Office procedure as practical action: Models of work and system design. *ACM Transactions on Office Information Systems*, 1(4):320–328, October 1983.

[27] Harrick M. Vin, Polle T. Zellweger, Daniel C. Swinehart, and P. Venkat Rangan. Multimedia conferencing in the etherphone environment. *IEEE Computer*, 24(10):237—268, October 1991.