1-7-2004

A Prototype System for Collaborative Virtual Environments: Design and Characterization

Moad Y. Mowafi

Thomas P. Caudell

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SCHOOL OF ENGINEERING
UNIVERSITY OF NEW MEXICO

A Prototype System for Collaborative Virtual Environments: Design and Characterization

Moad Y. Mowafi [1]
Department of Electrical and Computer Engineering
The University of New Mexico
Albuquerque, NM 87131
e-mail: `mowafi@ece.unm.edu`

Thomas P. Caudell [2]
Department of Electrical and Computer Engineering
The University of New Mexico
Albuquerque, NM 87131
e-mail: `tpc@ece.unm.edu`

UNM Technical Report: EECE-TR-04-xxx

Report Date: January 7, 2004

[1]Supported by the Office for the Advancement of Telehealth, Health Resources and Services Administration, Department of Health and Human Services.

[2]Supported by the Office for the Advancement of Telehealth, Health Resources and Services Administration, Department of Health and Human Services.

Abstract

Collaborative Virtual Environments (CVEs) are distributed virtual reality systems that allow multiple geographically distant users to communicate with each other and interact with virtual objects in a shared virtual world. Recently, there has been a growing interest in the development and use of CVEs. The design analysis of CVE systems reveals several critical technical issues: managing consistent distributed information, controlling access to objects by multiple users, supporting large number of users, handling new users joining an existing shared environment, and representing users to each other in the environment appropriately. As a result, various CVE systems have been developed to address these issues. Most existing systems are either focused on specific collaboration features or customized to particular range of target applications, which constrains the flexibility of the system architecture to support a wider range of applications and limits its extensibility to respond to new requirements. This document presents a design framework that addresses all of these major issues associated with CVE design and implementation, and a flexible open prototype implementation of this framework that explicitly exposes each of them, making evaluation of the effects of specific issues on overall system performance possible. A systematic approach to evaluation of whole system is proposed and applied through a focus on internal system characterization and a focus on the external effects of specific issues on system performance. The experimental results showed that the prototype system provides efficient performance for collaborative applications with respect to the requirements considered in the experiments. The evaluation of our prototype implementation shows that our approach is feasible, and can be applied for the evaluation of other instances of issues.

Keywords

Collaborative Virtual Environments (CVEs), modularity, consistency, concurrency control, scalability, late-join.

1 Introduction

A Virtual Environment (VE) is a computer system that generates a three-dimensional graphical virtual world, where the user experiences a sense of presence or immersion, and is able to navigate through this world and interact with graphical objects in it. Collaborative Virtual Environments (CVEs) are special cases of VE systems where the emphasis is on collaboration between users rather than on stand-alone or single-user applications. CVEs allow multiple independent instances of users to be immersed in a common or shared virtual world and have mutual awareness of each other's presence and actions. Users interact with each other and with objects within a shared virtual environment through distribution of information over computer networks. CVEs have been an active research area in both academic and industrial fields for several years, and a number of systems have been developed that vary in networking solutions, number of users supported, and application scope [29, 38, 11, 34, 19].

Most existing systems are either focused on specific collaboration features or customized to particular range of target applications, which constrains the flexibility of the system architecture to support a wider range of applications and limits its extensibility to respond to new requirements. For example, in systems that focus on collaboration within small scale groups, the mechanisms to handle larger scales are removed from the core design of the system. On the other extreme, there are systems that focus on large scale combat simulations with thousands of participants. In such systems, the whole design is focused on supporting a large number of users. Table I shows several application domains (military training, collaborative design, social collaborative work, and multiplayer games) and the major issues related to these applications. The issues include: consistency (managing consistent distributed information), concurrency control (controlling access to objects by multiple users), latency and bandwidth (minimizing the effect of network latency and reducing bandwidth requirements), scalability (supporting a varying number of users from large to small), and late-join handling (handling new users joining late an existing shared environment). The check marks in the table indicate the issues related to each application.

Application	Consistency	Concurrency Control	Bandwidth & Latency	Scalability	Late-join Handling
Military Training			√	√	
Collaborative Design	√	√			√
Social Collab Work		√		√	√
Multiplayer Games	√		√	√	

Table I: Relationship between CVE applications and the main technical issues associated with their development.

While many of the current systems have shown excellent performance with respect to certain features targeted by the system, the whole system performance has not been evaluated. Having an evaluation methodology for the whole system performance is important in order to show how well the system performs with respect to other collaboration issues, and it helps optimize the system design and identify the weaknesses and strengths of the system compared to other systems.

This document presents a design framework that addresses all of the major issues associated with CVE design and implementation, and a flexible open prototype implementation of this framework that explicitly exposes each of them, making evaluation of the effects of specific issues on overall system performance possible. A systematic approach to evaluation of whole systems is proposed and applied through an internal focus on internal system characterization and an external focus on the effects of specific issues on system performance.

In the following we present an overview of CVEs and their applications, then discuss the main research issues that arise in the development of CVEs and review several CVE systems with their approaches to these issues.

1.1 Collaborative Virtual Environments

Collaborative Virtual Environments (CVEs) are distributed virtual reality systems in which multiple users can communicate with each other and interact with objects in a shared virtual world. The user within the virtual world is represented by a graphical embodiment called an avatar [6] with which other users can interact and observe. In CVEs users exchange data and information over a local or wide area network that connects their hosts. (We will refer in this document to these hosts as nodes.) Figure 1 shows a general CVE system and flow of events (requests for changes to the shared virtual world) between its nodes. If an event occurs at node 1, a message about that event (a circle labeled ”1”) is distributed to all other nodes in the system. CVEs are suitable for numerous collaborative applications [26, 29, 12], for example: military training and simulations, medical training, virtual teleconferencing, engineering and design, collaborative scientific and information visualization, teleshopping, and multiplayer games.

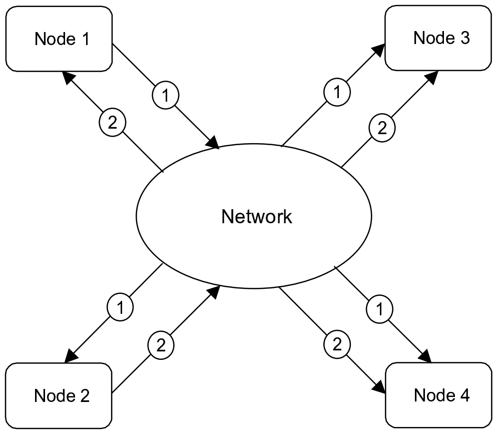


Figure 1: A general view of events and data flow between nodes in a CVE system. Arrows indicate event distribution.

1.2 Issues in Collaborative Virtual Environments

The design of CVE systems presents many research issues and technical challenges. The main issues associated with CVEs include: keeping the environment consistent among the users, controlling concurrent access to shared objects, network communication, supporting large number of users, distributing the shared world database, handling late-joining users, and user representation. In the following we discuss these issues and the range of current approaches to each of them.

Consistency

Consistency in CVEs commonly refers to the degree of matching among the users’ views of the shared virtual world. A high degree of consistency requires all events or actions issued by one user to occur at the same time for all users including the originator of the event. Network delay can cause inconsistency between users’ views of the shared world. For example, consider Figure 2. It shows a time-line diagram of a possible scenario for a two-event occurrence at two nodes in CVEs. Node 1 creates, executes and transmits event A (circle ”A”) at time t1 then node 2 creates, executes and transmits event B (circle ”B”) at time t2. Event A is then received and executed by node 2 at t3, and event B is later received and executed by node 1 at time t4. At this point, node 1 has executed event A then event B, while node 2 has executed event B then event A. The different ordering of event occurrence may result in inconsistent views of the shared world at both nodes. Various approaches have been adopted to maintain consistency such as synchronized delivery, total ordering, and source ordering [20].

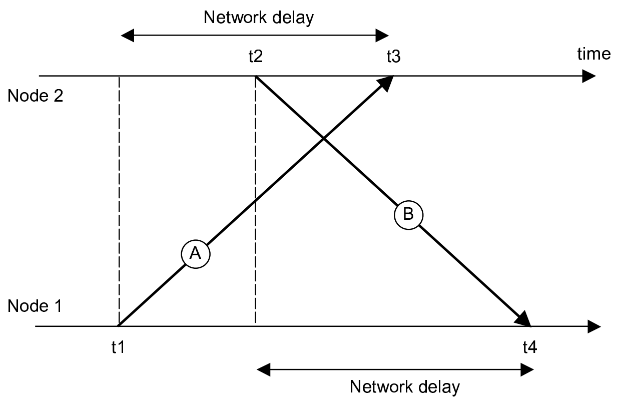


Figure 2: An example shows the effect of network delay on event occurrence at two nodes in a CVE system.

In the synchronized delivery approach, the events are delivered (or occur) simultaneously at all nodes including the node that originated the event. This requires a global clock or time-line to determine the delivery of events and the presentation to the users. In this approach, either all events are routed via a common server, which distributes the events to all nodes, or events are distributed to all nodes and a message must be received from every other node before the event can locally occur. This introduces a delay, between the event and its occurrence locally, of a round-trip time between the node that originates the event and the central server (or other nodes).

In the total ordering approach, a total order on all events in the CVE is applied, so that every node observes the same sequence of events in the same order. Determining a total order has essentially the same communication requirements as for synchronized delivery: either all events are routed via a common server which determines the total order, or a message must be received from every other node before the total order can be locally determined.

In the source ordering approach, events from each individual node are ordered by their source independently from events from other nodes. In this method, all nodes receive events from a particular node in the same order, while the order of events from different nodes is undefined. This form of event delivery is commonly provided by standard network protocols (like TCP/IP), and by reliable protocols.

Concurrency Control

In CVEs, when multiple users try to manipulate a shared object simultaneously, a certain control mechanism is needed to coordinate concurrent actions and prevent inconsistency. For example, if two users are moving a shared object to different places almost at the same time, then it is likely that, after both actions are performed, each user sees the shared object in a different location. Managing concurrency of such actions in CVEs is referred to as concurrency control. Most CVE systems adopt either pessimistic or optimistic concurrency control scheme.

Pessimistic schemes block a participating node from manipulating an object until an ownership request for that object is granted [16]. In this approach, ownership is granted by one of two means: either a request is sent to a central server that grants the ownership based on a policy, or is sent from the requesting node directly to the owner node in order to transfer the ownership. Pessimistic approaches are simple to implement and guarantee consistency, since there is only one owner at a time. Many existing CVE systems use pessimistic approaches [11, 25, 33, 37]. However, as the communication delay between the requesting node and the central server (or owner node) becomes high, the user interactions that involve transfer of ownership may be delayed.

Optimistic schemes allow participating nodes to manipulate objects without checking conflicts with other nodes. However, when conflicts occur, a repair must be performed [16, 36]. For example, a user may grab and move an object without checking for concurrent actions on that object from other users. If conflicting grabs occur, then resolving the conflict may involve undoing changes made to the object state (position, orientation). The advantage of this method is that, unlike the pessimistic approach, it avoids delaying the user interactions that involve transfer of ownership. On the other hand, when conflicts are repaired, users may get confused due to undoing previous actions or to sudden changes in the environment.

Network Communication

One of the critical components that supports CVEs is the computer network, a medium to exchange data and information among the multiple nodes participating in the shared environment. Common schemes used to exchange data between multiple nodes over the network are: unicast (point-to-point) and multicast (point-to-multipoint). In unicast, each node needs to send messages directly to all other nodes (see Figure 3a). For N nodes a total of $O(N^2)$ connections are established, which produces a large amount of network traffic. In multicast, a node can send a single message to a group of nodes at once, decreasing the number of connections for distributing a single message to N nodes from $O(N)$ to $O(1)$, which reduces the network traffic. (see Figure 3b). IP Multicast [31] is an Internet protocol, which allows IP datagrams (packets) to be sent to a group of recipients rather than a single node. To achieve this, datagrams are sent to one of the special IP addresses representing multicast groups. Nodes can join these groups in order to receive multicast packets. The packets are distributed via a subnet of the Internet called the MBone (Multicast Backbone), which consists of multicast routers responsible for replicating and distributing the multicast packet to its various destinations [24].

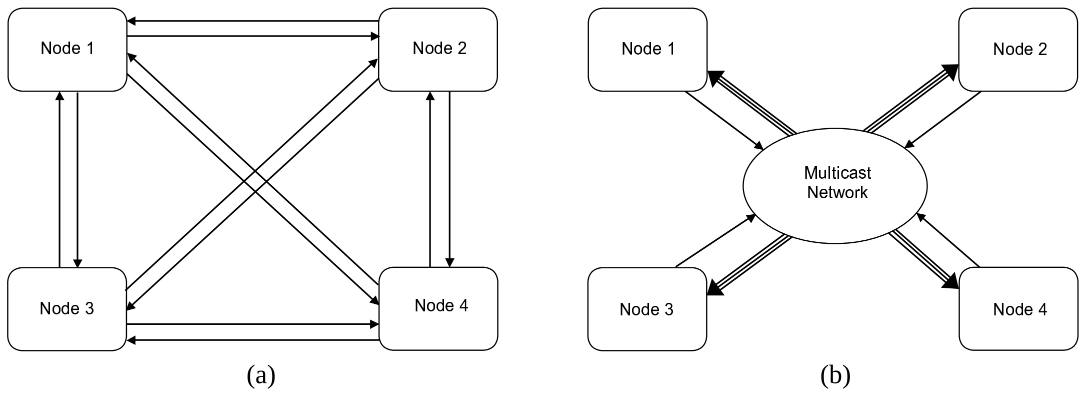


Figure 3: Network communication between multiple nodes using (a) unicast and (b) multicast connections.

The presence of a network in CVEs brings up a number of issues that determine their design and implementation. In the following we discuss three primary issues in network communication: bandwidth, latency, and reliability.

Network Bandwidth. Network bandwidth is the rate at which the network can deliver data to the destination node. The network bandwidth limitations pose a primary limitation for building large-scale CVEs. Therefore, many techniques have been developed to reduce the amount of communication. Methods that are applicable and popular in CVEs include: (1) exchanging command or event messages to update object states (position, orientation, or behavior) instead of distributing the updates themselves, (2) differential update messages, and (3) interest management [29, 11, 1, 17]. A differential message is an update message that instead of carrying the entire information about an object's current state, it carries only the difference between the current and previous state. Differential messages usually depend on a previously transmitted (reference) update message. In such cases the issue of reliability for those reference updates becomes important, where losing one of those reference updates causes the differential updates to create inconsistencies. In interest management techniques, nodes in CVEs express interest in certain objects, and only updates about these objects are received by those nodes. The node can locally discard unwanted messages, but this reduces only the local processing load and does not affect bandwidth consumption.

Network Latency. Network latency, or network delay, is the amount of time required to transfer a bit of data from one node to another. When a CVE application generates data for transmission over the network, the network latency determines when the application on the receiving node actually sees the transmitted data. This delay represents one of the biggest challenges to CVE development, because latency directly impacts the realism of CVE as perceived by the user. Many techniques have been investigated in order to reduce the source of latency or ameliorate its effects. The primary source of latency is usually network overload, and thus techniques have been developed to reduce the number and size of communication messages. IP Multicast can reduce network usage by delivering a single message to multiple nodes while ensuring that only a single copy of a message travels any shared network connections. All of the techniques mentioned previously as useful for reducing bandwidth consumption are applicable to reducing latency, since network congestion is a prime contributor to latency. To ameliorate latency effects, dead-reckoning techniques have been developed [32]. The idea is to transmit state update messages about an object in the virtual world less frequently and instead use information contained in those updates to approximate the true shared state. In between updates, each node predicts the position, velocity, acceleration, and other states of the object based on locally cached information.

Network Reliability. Network reliability is a measure of how much data is lost by the network during travel from source to destination node. Both reliable and unreliable protocols of unicast and multicast transmission are needed to optimally transport different classes of data such as object state information, user interactions, streamed audio/video feeds, geometric models, large scientific datasets, and tracker data. A tracker is a multiple degree of freedom measurement device that can, in real-time, monitor the position and/or orientation of multiple receiver devices in space, relative to a transmitter device. In VE trackers are used to locate hand held wands and to track the position of the user's head.

Unreliable protocols are suitable for the transmission of frequent data where the loss of a data packet is usually followed shortly by newer ones as in tracker data transmission and streamed audio/video feeds. Reliable transmission is important for the delivery of accurate state information as well as models and scientific datasets. Multicast protocols are based on unreliable UDP. To support reliable multicast, several reliable multicast protocols have been developed and used in CVEs [27].

Scalability

Scalability, in general, is a relationship between a certain measure of system performance (bandwidth, latency) and an independent variable which is a parameter of the CVE. In CVEs the parameter may be the number of users, number of objects, number of nodes, or physical distance between nodes. For example, a system is scalable if its bandwidth consumption increases sublinearly with the number of objects or nodes in the system.

Scalability can be hindered by various factors such as network bandwidth limitations and centralization of communication. The disadvantage of centralization of communication is that a single node (server) must handle all of the message traffic. Multiple server architectures have been used by several systems to overcome part of the message centralization problem [14]. Multicast is another approach that has been used to avoid the centralization of communication [29, 11]. Network bandwidth can limit the scalability of CVEs. As the number of nodes, say N, increases, the potential communication increases with $O(N^2)$, because each node could potentially communicate with every other node in the CVE. Approaches to the problem of network bandwidth limits were discussed previously in the network communication issues.

Data Architecture

The VE database represents the data that is relevant to the state of the virtual world and its objects. It contains information that is used to create the environment (e.g., terrain, structures) and its objects. CVE database is usually partially or totally replicated on each participating node to reduce the amount of communication. Using this technique, only the updates or changes of a shared state need to be sent to the participating nodes. The data architecture in CVEs can be divided into three main classes: replicated, centralized, and distributed [28].

Replicated Architecture. In this architecture, each node holds a completely replicated database of the shared environment, and state information is shared by broadcasting messages to all participating nodes (Figure 4). This architecture has no centralized control; hence any new node joining a collaboration session gathers state information about the world from the other nodes.

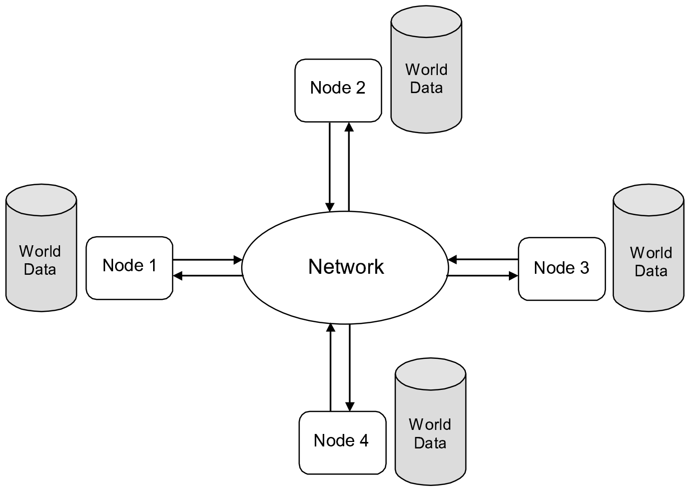


Figure 4: Replicated data architecture.

Centralized Architecture. In this architecture, the data of the shared environment is stored at a central server (Figure 5). The main advantage of this scheme is that it greatly simplifies the management of multiple nodes, because the central server coordinates all requests issued by multiple nodes. However, the role of the server as an intermediary for the delivery of data can impose an additional delay in the system. Another disadvantage is that if the central server fails, none of the connected nodes can interact with each other. Despite these disadvantages, this architecture is still useful for supporting small groups of collaborators.

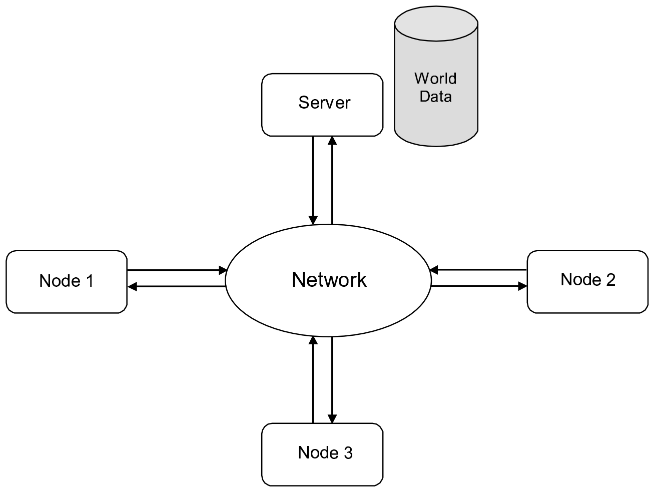


Figure 5: Centralized data architecture.

Distributed Architecture. In this architecture, the world database is distributed either among the existing nodes or among multiple servers. Distributing the database among the existing nodes (Figure 6) simulates a wide-area shared memory structure in which objects that are instantiated at one node are automatically replicated at all the remote nodes. Typically in this architecture, a newly connected node must form point-to-point connections with all the existing nodes. Hence, for N nodes the number of connections required is $O(N^2)$. In addition if the environment involves the sharing of large datasets, the dataset will be fully replicated at every node. Unless the data sharing policy is modified to account for large datasets, this architecture will not be scalable in the number of nodes. Distributing the world database among multiple servers allows the participating nodes to connect to different servers and obtain the needed data. A classic approach is to bind the servers to unique multicast addresses. Nodes then subscribe to different multicast addresses to listen to broadcasts from the servers [1, 14]. This is a particularly effective way to handle large numbers of connected nodes distributed over a wide virtual world. Each geographic region of the virtual world can then be maintained by a separate server.

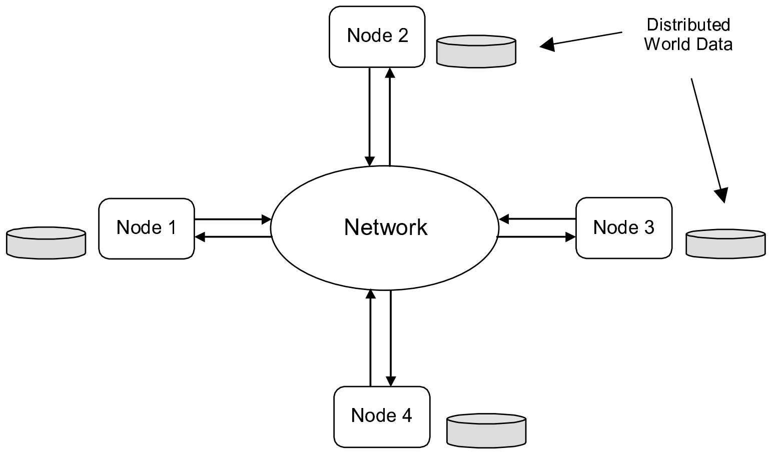


Figure 6: Distributed data architecture.

Late-Join Handling

When a new node (or user) joins a collaboration session, it needs to obtain the current state of the virtual world. One common approach to handle late-joining nodes is to have one of the existing nodes in the session sends the current state of the world to the late-joining node [29]. The choice of which node performs this task is important because, if the same node is chosen every time, and new nodes join frequently, then the node will be overburdened with requests for the current state. This approach has a potential problem of hindering performance on the node that is providing the current state of the world. Another approach is to have a central server that provides late-joining nodes with the current state of the world [11, 1, 35]. With this approach, the system will encounter scalability problems but will avoid impacting the performance of its existing nodes.

User Representation

User representation determines the way users observe each other's actions in the virtual world. Users' embodiments, or avatars, replicate the actions taken by the real users located in remote environments. The geometric complexity of avatars can vary according to the task being performed. In most existing systems, avatars range from primitive cube-like appearances [18], non-articulated human-like or cartoon-like avatars [2] to articulated body representations using rigid body segments [1, 11]. Different rendering techniques for avatars have been used in CVEs. Video texturing is used to place texture mapped, live video images on the head of a 3D model of a human. While this technique provides realistic facial expressions, it requires high bandwidth to transmit video streams. Complex modeling is another technique that aims at high level of realism in CVEs. The main issues with complex modeling are the rendering speed and network overload because of the huge amount of data that is needed to achieve a high level of graphical detail and natural interaction between the users within the environment.

The above discussion has addressed several issues in CVE development and briefly discussed the current approaches to these issues, as well the issues that these approaches raise. Next, we review a number of CVE systems and the approaches adopted by these systems with respect to the issues discussed above.

1.3 CVE Systems

To date, there have been significant research efforts on CVEs. As a result, many systems have been developed that differ largely in networking solutions, number of users supported, and application scope. In the following we summarize several features of a set of CVE systems that have contributed to the field of CVEs, and their approaches to the issues discussed in the previous subsection.

NPSNET

NPSNET, developed at Naval Postgraduate School (NPS), is intended to be used by the US Department of Defense. NPSNET-IV [29] was designed to comply with DIS protocol. DIS (Distributed Interactive Simulation) [8] is a standard, which focuses on military simulations. DIS utilizes a distributed architecture with no central server. The objects interact by broadcasting events to the network, and the receiving node is then responsible for calculating the effects. Between the updates, the object's position information is extrapolated by dead reckoning. DIS requires all objects to send periodic update messages. A new joining node has to listen for a while until the next refresh cycle of updates happens. NPSNET-IV included enhancements to DIS such as the use of IP Multicast with dynamic multicast groups reflecting the partition of the virtual world.

The latest version, NPSNET-V [7, 22], is being designed to become a platform for research on infrastructure for Large Scale CVE. It is written in Java and uses the HLA standard for communication. HLA (High Level Architecture) [23] software architecture aims at providing a general architecture and services for the development of component-based simulations. It provides a set of common services that are useful across multiple simulation domains. The services include: object management; ownership management; time management; and data distribution management. NPSNET-V goals include dynamic simulation component loading, dynamic object behavior modification, and dynamic protocol modification.

DIVE

DIVE (Distributed Interactive Virtual Environment) [11], developed by the Swedish Institute of Computer Science, uses a replicated world database. DIVE compensates inconsistencies due to network delays by employing dead reckoning and by sending periodically synchronization information. DIVE uses IP Multicast for audio and video, TCP for state transfer to new joiners and the Scalable Reliable Multicast (SRM) approach for state updates. To reduce network traffic DIVE allows the VE to be partitioned into sub-hierarchies (e.g., based on geography) that are replicated among a smaller set of nodes that have expressed interest in them.

SPLINE

SPLINE (Scalable PLatform for Interactive Environments) [1], developed by Mitsubishi Electric Research Labs, is a software platform that allows the development of real-time multi-user social cooperative applications. In SPLINE the VE is divided into locales (regions), which have arbitrary size and shape and are associated with multicast addresses. Each node maintains a partial copy of the VE corresponding to the focus of attention. SPLINE uses the Interactive Sharing Transfer Protocol (ISTP) [37], which is a hybrid protocol that builds on top of four underlying protocols: TCP, UDP, HTTP, and RTP (Real-time Transport Protocol). ISTP contains five sub-protocols: One-to-One Connection Subprotocol to establish and maintain a TCP connection between two ISTP nodes; Object State Transmission Subprotocol to communicate the state of objects from one ISTP node to another via point-to-point connection or UDP Multicast; Streaming Audio Subprotocol to stream audio data via RTP; Locale-Based Communications Subprotocol to support the sharing of information about objects in the virtual world; and Content-Based Communication Subprotocol to support central server style communication of beacon information.

MASSIVE

MASSIVE (Model, Architecture and System for Spatial Interaction in Virtual Environments) [17], developed by the University of Nottingham, allows users to interact with each other over a combination of graphics, audio and text media. MASSIVE has three generations. The latest one, MASSIVE-3 [19], uses a distributed world database. Each distributed database (called an Environment) describes one well-defined portion of the virtual world. Networking is logically multicast, but physically client-server, using unicast network protocols (TCP/IP). Every joining node contacts a single master node for a given Environment. The master node for each Environment also acts as a network server, forwarding events to all nodes currently replicating the Environment. Consistency is preserved by various methods of updating, such as transferable ownership (only the owner of an object can manipulate it) and centralized update (every update of object state is sent to the owner of the object to perform the update).

VLNET

VLNET (Virtual Life Network) [5], developed by EPFL Computer Graphics Lab and University of Geneva, is a client-server system. The server uses the HTTP protocol to distribute database information to the nodes and to manage sessions. VLNET distributes its virtual worlds in its proprietary file format or as VRML (Virtual Reality Modeling Language) files. VLNET incorporates highly realistic virtual human representations. The users are represented by 3D virtual human actors, which serve as agents to interact with the environment and other agents. The agents have similar appearance and behaviors as for real humans.

The remainder of this document is organized as follows. In Section 2 we present our approach to the issues listed in this section, the architecture design of our CVE system, and our methodology to performance evaluation. In Section 3 we present the results of experiments with the prototype of our system for performance evaluation. A discussion of the results is presented in Section 4. Conclusions and future work are presented in Section 5.

2 Design and Evaluation Approach

The goal of this research is to develop a design framework that addresses all of the major issues associated with CVE design and implementation, and to implement a flexible open prototype of this framework that explicitly exposes each of them in order to evaluate the effects of specific issues on overall system performance. In this section we describe our approach to design a flexible CVE system framework, the architecture design of our CVE system, and our methodology to performance evaluation.

2.1 Design Framework

In our approach to developing a framework that addresses all of the design issues presented in Section 1, we use *modularity* as a key factor in the framework development. In the following, we first explain modularity in this context and then describe the design framework.

Modularity

System architectures can be monolithic, built for a special purpose or a particular range of applications, or modular, composed of separate components (modules) that can be combined together to provide specific application functionality. While monolithic architectures can provide high performance, they are hard to modify or extend to support new or different applications. In modular systems, modules can be modified and extended without affecting other parts of the system, providing flexibility to add new features to the system and to adapt to new application requirements.

In order to design a flexible CVE system that can deal with the design issues presented in Section 1, we adopt modularity in our approach to achieve this goal. In the following we present the key elements of our approach to developing a framework for flexible CVE systems.

Collaboration Module

The modularity of a system allows a common set of functions to be implemented as a self-contained module. Collaboration services (such as data distribution, object ownership management, and handling new joining users) can be implemented as a self-contained collaboration module. This allows development to focus on the application itself separating the specific functionality from the collaboration tasks. Thus, applications developed for a single user can be shared in the same manner as those developed for multiple users. As a result, users can run their applications in a single-user mode or share them through the collaboration module. Another advantage of modularity is that the CVE implementation is focused on a particular module rather than the whole system, and new techniques can be incorporated without having to change the design core of the system.

Collaboration Threads

To provide flexibility in handling and managing shared information, the collaboration services can be applied to different types of data and events in multiple ways, for example, applying different levels of reliability for data delivery or different forms of consistency in events occurrence, according to the application requirements. The collaboration module functionality should be implemented using multiple threads to provide such flexibility, where different threads can handle different types of data and events. In addition, new types of data and events can be added by adding a new thread associated with that type of data or events, providing greater flexibility. Moreover, using multiple threads allows concurrent processing of incoming and outgoing messages, which optimizes the system performance.

Event Interception

In order to provide support for both collaboration-aware applications (explicitly developed for multiple users) and collaboration-unaware applications (developed for a single user), a collaboration transparent mechanism is needed. Events generated by a user or system can be intercepted before being executed at the system core level without the need for a knowledge of how these events are generated at the application level. Consequently, events generated from collaboration aware or unaware applications should be treated similarly. This can be achieved by developing a wrapper code (in-line code or a call to a given function) at the system core level. The wrapper checks whether the generated event is private or shared. Private events are processed without interception, while shared events are queued into global queues to be distributed to remote applications by collaboration module sending threads. At remote nodes, collaboration threads receive and prepare the events for execution (see Figure 7). In real-time applications, it is inefficient for the collaboration threads to poll for shared events. Instead the wrapper notifies the responsible thread when the event arises, using wait-signal mechanism. In this method, collaboration threads are blocked waiting for a signal to be generated by the wrappers. Mutual exclusion is used along with this

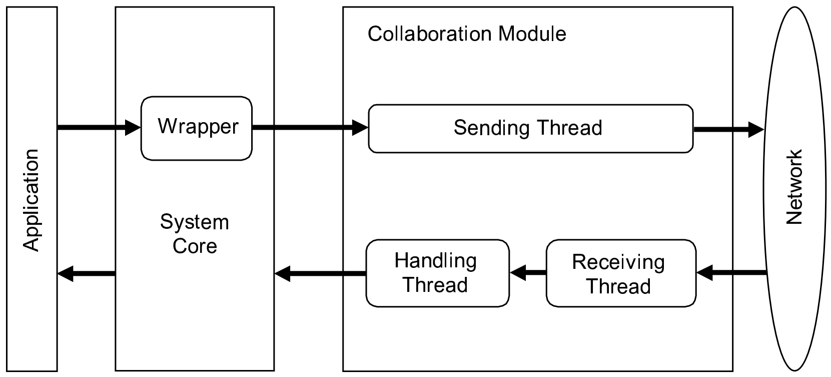


Figure 7: Event interception. Events (arrows) are intercepted by wrappers at the system core and queued to be distributed to remote applications by collaboration module sending threads.

mechanism to provide the necessary locking when accessing global queues. Mutual exclusion is a mechanism to ensure that only one thread (or process) can access a shared resource at a time.

Symmetric Events and Data Distribution

Using the event interception mechanism through wrappers means that local events are not executed immediately. Instead events are queued to be managed by the collaboration module. The collaboration module then distributes the events to every node in the environment, including the node that originated the events (see Figure 8). In this model, events and data are symmetrically distributed to both sender (local node) and receiver (remote node), which causes the lag between local and remote updates to object states to be determined by the networking resources such as latency and bandwidth.

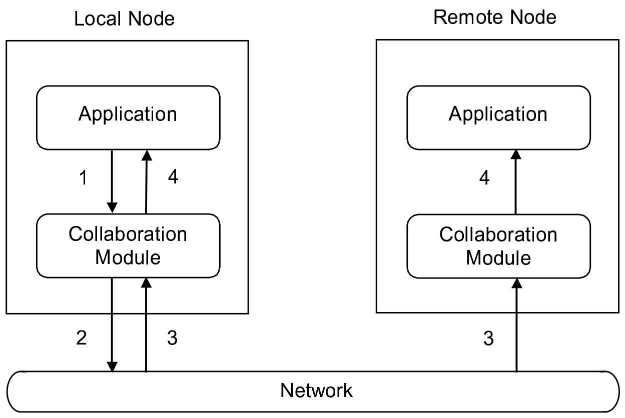


Figure 8: Symmetric events and data distribution.

The collaboration module, event interception, and symmetric events distribution, all together, provide a flexible framework that can address all of the major design issues in CVEs. The collaboration module provides flexibility to adopt different solutions and approaches to the issues of interest. The event interception and symmetric events distribution provide transparent collaboration, making the deployment of different mechanisms and techniques to support a wide range of applications transparent to these applications.

We have presented our approach for a framework that provides support for developing a flexible CVE system that can deal with the design issues presented in Section 1. Next, we describe the architecture design of our prototype system that implements this framework.

2.2 Prototype

To develop a CVE system based on the framework described in the previous subsection, the system architecture should support modularity and openness. We have implemented a flexible open prototype using a virtual reality application development environment, called Flatland, which provides an ideal infrastructure that meets the above requirements. In the following we first introduce Flatland environment and briefly discuss its components, and then present a detailed description of the architecture design of our prototype system.

Flatland

Flatland is an open visualization/virtual reality application development environment (see Figure 9). It allows software authors to construct and users to interact with arbitrarily complex graphical and aural representations of data. Flatland is multithreaded and uses dynamically linked shared libraries to assemble modules that construct or modify the virtual environment for a specific application. A Flatland application consists of the Flatland core and an arbitrary number of modules. The core is a set of basic routines and services that provide Flatland its basic functionality. Modules add application specific functionality to the core, and are dynamically loaded and linked to create applications. Modules may be loaded on top of other modules to incrementally build complex functionality (see Figure 10). In the following we describe the main components of Flatland and its architectural features.

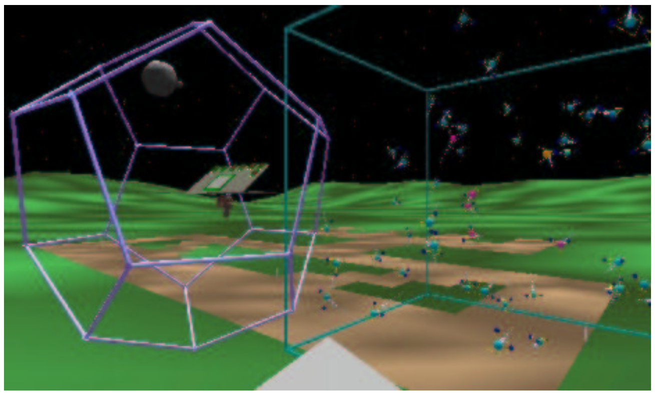


Figure 9: A view from within a typical Flatland environment. The dodecahedron is the personal vessel of the user. The sphere is the location of the camera (or eye) through which the virtual world is viewed.

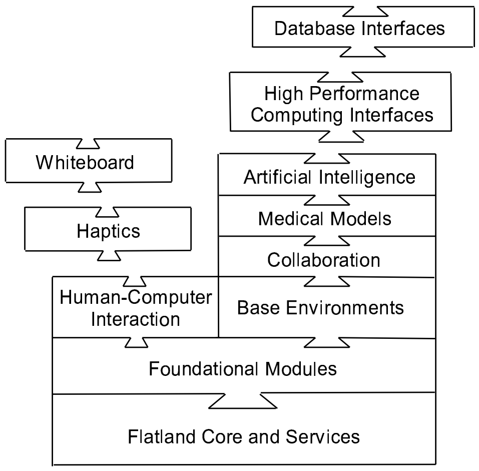


Figure 10: An illustration of the modular nature of a Flatland application.

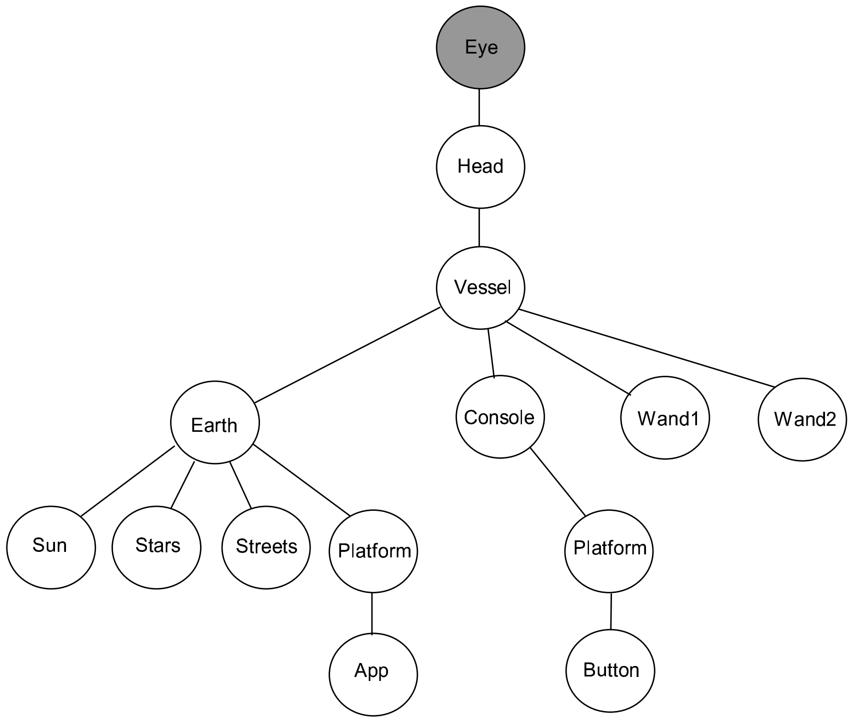


Figure 11: An example Graph configured as a tree with the ”Eye” Object as the root. The darkened circle represents the current root.

Flatland Transformation Graph. A Flatland world is a hierarchical database of what is called Objects. Flatland Transformation Graph is a global graph data structure that stores all of the geometric properties that interrelate the relative positions and orientations of all Flatland Objects, which are represented as vertices (Figure 11). The Graph can be configured into different trees, with one Object acting as the root. It is from the root Object's perspective that the virtual world is viewed. The Graph Objects are connected through links, which contain the information about the relative geometrical transformation between neighboring Objects. This information is stored in a data structure called Mpoparam. The Mpoparam structure contains inter-object transformational parameters, a transformation program, and a resultant 4x4 transform matrix, Mpo, or matrix parent-object. The Mpo matrix transforms points in the Object's model coordinate frame into its parent's coordinate frame. To compose the Mpo matrix, the transformation program specifies the order in which the transform operations (translation, rotation and scale) are applied to the transformational parameters.

Flatland Objects. A Flatland Object contains data and functions. The functions provide all of the behaviors of an Object, giving it the ability, for example, to draw itself, to make sounds, to draw its shadows, and to manage its placement onto other Objects when it is created. The Object's data consist of general purpose variables and pointers, a set of bounding volumes, a set of controls, and a 4x4 transform matrix, Mco or matrix camera-object. The Mco matrix contains the transformation from the Object's model coordinate system to the root Object's model coordinate system. Bounding volumes are used for determining the relative Object's position with respect to its parent, graphical culling and collision detection. The Object's controls provide different methods for user interaction. The controls can be evoked through the keyboard, menu or voice recognition. Moreover, Objects can be controlled by external systems like simulations and trackers (devices to track the user's head and hands).

Flatland Modules. A module in the context of Flatland is a relatively self contained collection of Objects, functions and data that can be dynamically loaded (and unloaded) into the Graph of an environment. The module is responsible for creating and attaching its Objects to the Graph, and for supplying all Object functionality. A module is added to Flatland through the use of a configuration file. This file contains a structured list of modules and their user specifiable parameters.

Flatland Threads. Flatland is intrinsically multithreaded, allowing the system to make use of computer systems with multiprocessors and shared memory. The Flatland kernel has one main thread that spawns three child threads: the graphics, sound, and tracker threads (Figure 12). The graphics thread is responsible for graphics rendering, which includes culling, collision detection, lighting and displaying Flatland Objects, their shadows and transparent portions. The sound thread is very similar to the graphics thread; it is responsible for sound rendering. The tracker thread communicates with any tracking devices that may be attached to Flatland. The tracking devices include two wands and a head tracker. In addition to these threads, each module in Flatland can spawn its own threads independently.

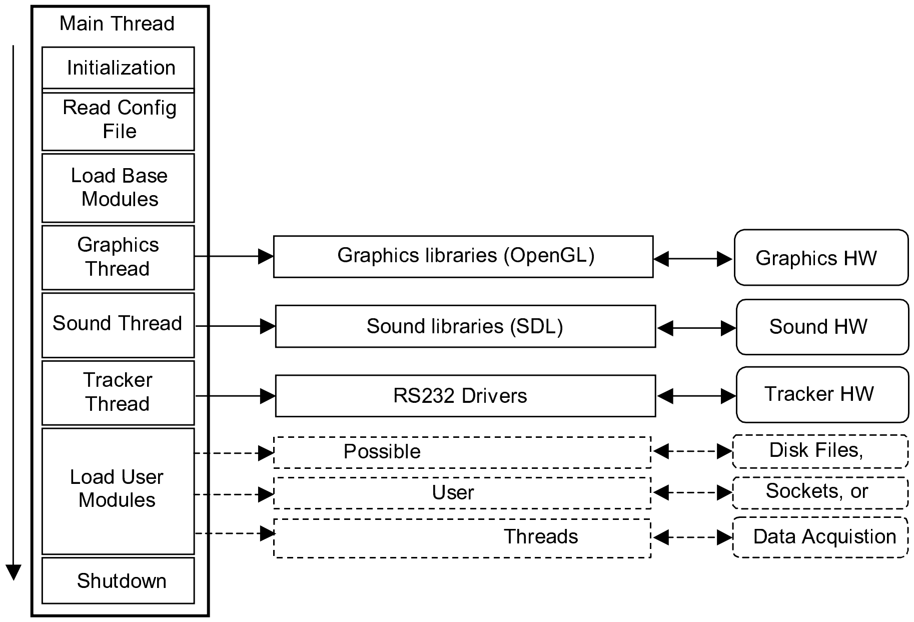


Figure 12: An overview of the main threads in Flatland. The ”main” thread performs initialization, reads the configuration file, and spawns the three support threads. The main thread may also spawn other threads defined by the application designer.

Architecture Design

In the following we present a detailed description of the architecture design of Flatland collaborative system. An overview of the CVE system architecture is presented first, followed by detailed descriptions of the collaboration module and the services provided by the module.

Design Overview. Flatland collaborative environment uses a replicated world database. Each node contains a copy of the applications that are to be collaborated upon (Figure 13). The information is exchanged among participating nodes through a kind of group communication mechanism. When a node updates its Object’s state, all other nodes are notified and receive the updated state. The exchange of update messages is implemented using IP Multicast to reduce the network load and increase scalability. To reduce the amount of communication, state updates that depend on the passage of time and are independent of user input (such as the animation of a moving Object) are executed locally and are not transmitted at all since each node can independently calculate the

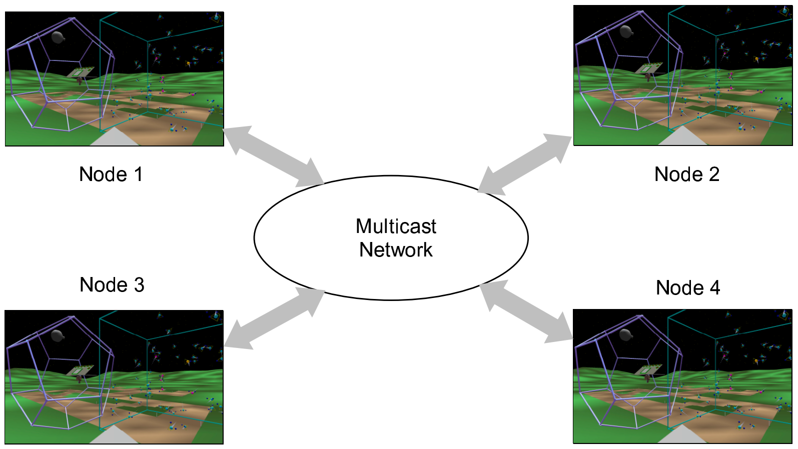


Figure 13: An overview of the architecture design of Flatland collaborative environment.

required state updates. In addition to multicast, the system provides support for unicast communication between two nodes.

Collaboration Module. Collaboration services such as event and data distribution, object ownership management, and handling new joining users are implemented in a self-contained collaboration module. The collaboration module provides the following services:

- Events and Data Distribution: keeps distributed copies of a shared Object consistent.
- Ownership Management: manages the ownership of shared Objects to prevent simultaneous actions by multiple users.
- Late-Join Handling: provides new joining nodes with the current state of the shared world.
- Avatar Management: handles the entry and exit of users' avatars and keeps their states up-to-date.

In the following we discuss the implementation of each of these services.

Event and Data Distribution. The collaboration module multicasts update messages to the remote nodes as well as local node. In order to provide an acceptable response to local updates, the collaboration module benefits from the loop-back feature in IP Multicast, where a copy of the multicast message is looped back by the IP layer for local delivery.

The collaboration module functionalities are split into several processes implemented as threads. These collaboration threads are independent (asynchronous) of Flatland main threads (graphics, sound and tracker). The collaboration threads essentially deal with incoming and outgoing events. On the sending side, each collaboration thread is responsible for extracting the event data from the sending queue filled by the associated wrapper, managing ownership, packing data into a network message and transmitting it over the network. Queuing and dequeuing data is managed by the wait-signal mechanism, described previously. On the receiving side, a receiving thread receives network messages and distributes them to different threads based on the message type (discussed later). This requires minimal examination of message contents before the message is distributed to the event-handling thread. Each event-handling thread is responsible for unpacking the event data and executing the event. The distribution of received messages to different threads is implemented using the wait-signal mechanism. The receiving thread queues the received messages into receiving queues and notifies the responsible thread, which extracts the messages and processes it. The multithreading model allows concurrent processing of different events either on the sending or receiving side.

A communication message has a standard header that determines the sender and the message type, and the message body. The message body content depends on the message type. Figure 14 shows a classification of the communication messages used by the system. The messages are classified into three main types: session messages, module messages, and Object messages.

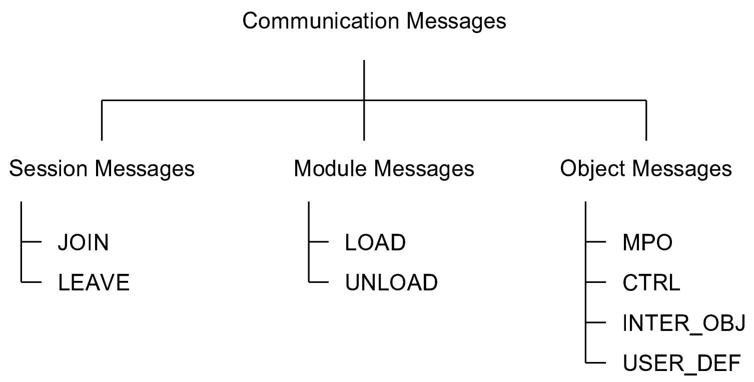


Figure 14: Communication message types in the CVE prototype system.

Session messages (JOIN and LEAVE) are used to establish and terminate connections between collaborative nodes. The session messages contain a user identifier (logname, for instance), which is associated with the user's avatar during the session.

Module messages (LOAD and UNLOAD) are used to load and unload Flatland modules. The module messages contain the name of the module being loaded or unloaded.

Object messages (MPO, CTRL, INTER_OBJ and USER_DEF) are used by Flatland Objects. The MPO message updates the Mpo transformation matrix of an Object. The MPO message contains the Object's name, its parent's name, parent-object transformational parameters and transformation operations. The CTRL message is used for an Object to send a command key, which evokes the Object's control function associated with the key. The CTRL message contains the Object name and a command key. The INTER_OBJ message is a general-purpose message used to exchange a string message between two Objects. The INTER_OBJ message contains the sender Object's name, the receiver Object's name and a string message. The USER_DEF message is a user-defined message designed to be used in high-performance computing applications. For example, simulation calculations can be performed on a high-speed node, which broadcasts the results to the other nodes for visualization.

The size of the communication message depends on the message type. All of the messages except INTER_OBJ and USER_DEF messages have a fixed size between 100 and 260 bytes. The size of INTER_OBJ and USER_DEF messages depends on the user-defined data contained in the message.

Ownership Management. Typically, concurrency control approaches may be either pessimistic to avoid inconsistency in the shared environment, or optimistic, not imposing restrictions on user interactions and usually attempt to resolve the inconsistencies when they arise. In our prototype, a conservative (pessimistic) scheme based on a single transferable ownership for each Object is used. At any time at most one node is the owner of an Object, and only this node is allowed to update the state the Object or call its control functions. This is implemented using a central server called the Lock Manager, which manages ownership of shared Objects. Participating nodes contact the Lock Manager to obtain or release ownership. The Lock Manager ensures that there is one owner at time for each shared Object. Initially, the Object is owned by the node that created it. The TCP protocol is used to exchange messages between nodes requesting ownership and the Lock Manager; TCP is reliable to guarantee ownership transfer.

A node owning a shared Object can give away ownership and, on the other hand, any other node can try to acquire ownership of that Object. The Lock Manager's response to requests for ownership is based on the lock status of the Object of interest and the lock policy. Ownership of an unlocked Object will be given in response to any request, while ownership request of a locked Object will be rejected (Figure 15).

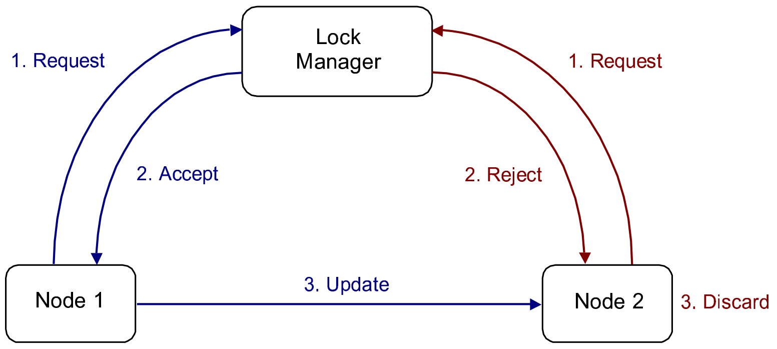


Figure 15: An example of ownership management in the CVE prototype system. The Lock manager grants the ownership request from Node 1, while denies the request from Node 2, based on the lock policy.

Ownership requests are performed by the collaboration module. When a node generates an update for a shared Object, the collaboration module checks the ownership of the Object before transmitting the update. If the node does not own the Object, then the ownership of the Object is requested. If the request is satisfied then the update is transmitted; otherwise the attempted update is discarded. Subsequently, as long as the node retains the ownership of an Object, its updates will be transmitted immediately with no requests. This approach drastically reduces the inter-node communication because participating nodes receive updates only from the node that owns the Object. Otherwise, every attempted update will be transmitted to every node even the updates issued by a node that does not have ownership. In this case, every node that receives the attempted update needs to check for ownership before accepting the update.

Late-Join Handling. CVE nodes can join and leave the collaboration session dynamically by loading and unloading the collaboration module. When the collaboration module is loaded it sends a JOIN message via a multicast address assigned to the virtual world, and all the nodes within the world receive the join message (Figure 16). When a node receives a JOIN message, it sends the current state of all the shared Objects created by that node, using a background collaboration thread. Consequently, the first messages the joining node receives are LOAD messages, which cause it to create copies of the Objects already present within the world when it joined. Subsequent updates (MPO messages) for these Objects are delivered after the initial transfer is complete.

With this approach, the burden of centralization of handling new joining nodes is partitioned over many nodes. However, when a node leaves the collaboration session it must destroy all the Objects it has created; otherwise these Objects will have no node to send their states as a response when a new node joins the session. To overcome this constraint, the Lock Manager can be used to assign a new creator to these Objects among the current nodes, or distribute the Objects among a number of new creators efficiently, for example, using load balancing algorithms.

Avatar Management. Each user in Flatland shared environment is represented by one Object (or more) whose state is controlled by user input and kept up-to-date on other users' environments through MPO messages. A user representation, or avatar, is implemented as a Flatland module that can be dynamically loaded when a user joins the environment, and unloaded when the user leaves. A personal configuration file, similar to Flatland configuration file, is provided so that users can customize their own virtual representation in the environment. This provides flexibility in designing the user avatar using either simple non-articulated or complex articulated models, according to the application requirements.

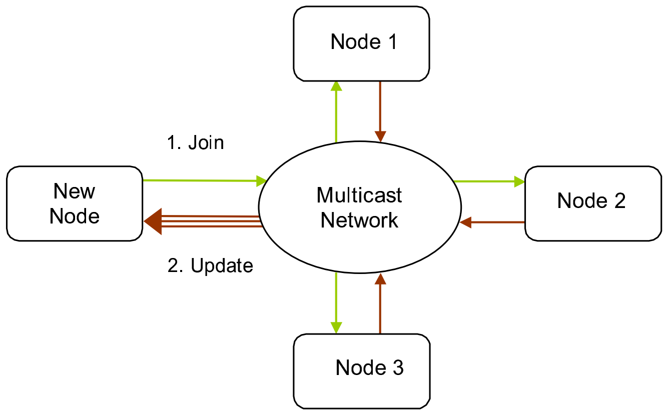


Figure 16: A distributed approach for late-join handling in the CVE prototype system. A new node obtains the current state of the world from the existing nodes.

In Flatland, trackers are used to locate hand held wands and to track the position of the user's head. Tracking the orientation and position of user's head and hands generates a huge amount of position and orientation update messages that are distributed over the network. To reduce the number of these messages, only transformational parameters of avatar's position and orientation that exceed a certain threshold are transmitted. The threshold value can be adjusted according to the application requirements.

We have described the architecture design of our system to support collaboration in virtual environments. Next, we present a systematic approach to performance evaluation.

2.3 Evaluation Approach

Our approach to performance evaluation involves two steps:

1. Characterizing the internal system design by local experiments in order to uncover any design flaws or refine the design approach, and give an initial evaluation of the system performance.
2. Evaluating the effects of the specific issues on overall system performance by external networked experiments.

In the following we describe our methodology to evaluate our system based on the above steps.

System Characterization

Characterizing the internal system design focuses on evaluation of performance and scalability of the collaboration model used by the system. In our collaboration model, local events generated by a local node are intercepted and are not executed immediately. To examine the effect of the collaboration model on the execution of local events, we study the effect of each operation performed during the path between event generation and execution. These operations include: queuing generated event; packing event data; looping back network message; queuing received event; and unpacking its data to be executed. To evaluate the performance of the collaboration model, we define the following performance parameters:

1. Generated Message Queuing Delay: The time it takes for a generated message to be queued by associated wrapper and dequeued by collaboration thread.

2. Generated Message Packing Delay: The time it takes to pack the data of a generated message into a network packet.
3. Loop-Back Delay: The time it takes to receive a looped back copy of a multicast message.
4. Received Message Queuing Delay: The time it takes for a received message to be queued by receiving thread and dequeued by collaboration thread.
5. Received Message Unpacking Delay: The time it takes to unpack the data from a received message.
6. Local Delay: The time interval between message generation and execution, locally. This delay includes all the above delays.

To examine how well our system scales, we evaluate the performance of the collaboration model in large-scale environments. Specifically, we examine the effect of the message throughput (number of received messages per second) on received message queuing delay at one node in a large-scale environment.

System Evaluation

To evaluate the effects of specific issues on overall system performance, we define a range of performance parameters associated with each issue. The issues being considered are: data distribution, ownership management and scalability.

Data Distribution Evaluation. Event and data distribution keeps distributed copies of the shared virtual world consistent. To evaluate our design approach to data distribution, we define the following performance parameters:

1. Throughput (or Bandwidth): The number of bits received per second at one node.
2. End-to-End Delay: The time (in milliseconds) it takes for a message at one node to be packed, sent over the network, received by another node, and unpacked to be processed. It does not include the rendering and graphics delays.

In our system, the end-to-end delay includes the local delay introduced by the collaboration model and network latency. Because update messages are symmetrically distributed to all nodes including the node that originated the update, both local and remote updates will experience the same local delay. This delay can be independently measured at the local node, and then combined with the network latency between the local and remote nodes in order to obtain the end-to-end delay.

Ownership Management Evaluation. In our system, only the owner of an Object is allowed to manipulate the Object. Ownership of an Object must first be acquired in order for a node to manipulate the Object. Obtaining the ownership requires one round-trip time from the node that requests ownership to the Lock Manager and back. This transfer of ownership creates an additional delay to the event execution process both at local and remote nodes. To evaluate our design approach to ownership management, we define the following performance parameters:

1. Ownership Transfer Delay: The time it takes for a user to obtain ownership of an Object.
2. Response Time: The time interval between a user's input action and its response seen by the same user.
3. User-to-User Time: The time interval between a user's input action and its effect seen by a remote user.

In our system, the response time includes the local delay introduced by the collaboration model and ownership transfer delay. The user-to-user time includes the end-to-end delay and ownership transfer delay. Ownership transfer delay can be measured independently of the local delay and end-to-end delay. The measured delay can be then combined with the local delay to obtain the response time, and with the end-to-end delay to obtain the user-to-user time.

Scalability Evaluation. To evaluate the scalability of our system, we study the relationship between the system throughput and the number of users simultaneously present in the shared environment. In our evaluation, we study the effect of use patterns (how users use the system to interact with the virtual environment) on system scalability. The use patterns can determine the degree of movement and degree of interaction among users and between users and objects. For instance, a user may spend more (or less) time moving rather than stationary, and may move fast around the virtual world generating traffic more than a user moves slowly. In addition, user movement and interaction can be coordinated to perform specific actions at the same time, which results in bursts in network traffic. To examine the effect of use patterns, we evaluate the system scalability under real-world scenarios. Specifically, we examine the scaling of throughput with the number of users having different use patterns.

This section has described the architecture design of our system and our performance evaluation methodology. Next section presents the results of experiments conducted for performance evaluation.

3 Experiments and Results

This section describes the experiments with the prototype system, and presents the results of measurements for performance evaluation. We used the performance evaluation methodology described in Section 2 to set up and perform these experiments. A discussion of the results will follow in the next section.

The experiments are divided into two types:

- Local experiments to characterize the internal system design.
- External networked experiments to evaluate the effects of specific issues on system performance.

The experiments were performed with an instrumented version of the system developed to collect performance measurements. Instrumentation points for timing events generation, transmission, reception and execution were inserted through routines integrated with the collaboration module. In the following we describe the local and external networked experiments, and present the results of performance measurements.

3.1 Local Experiments

In this subsection we present the results of local experiments performed to characterize the internal system design, specifically the collaboration model. We performed a set of experiments to measure the performance parameters defined in Section 2. The results of performance and scalability measurements are presented next.

Internal Performance Measurement

A series of local experiments were performed with the instrumented system. In each experiment, we measured the time between event (update message) generation and execution, on the same machine. Specifically, we measured the generated message queuing delay, generated message packing delay, loop-back delay, received message queuing delay, received message unpacking delay and local delay.

A Flatland simulation application was developed to perform the experiments. Figure 17 shows the message flow for a local event issued by a Flatland application, and instrumentation points used to measure the above delays. The generated message queuing delay is measured by the amount of time it takes for a message to travel from point (1) to point (2). The generated message packing delay is measured by the amount of time it takes for a message to travel from point (2) to point (3). The loop-back delay is measured by the amount of time it takes for a message to travel from point (4) to point (5). The received message queuing delay is measured by the amount of time it takes for a message to travel from point (6) to point (7). The received message unpacking delay is measured by the amount of time it takes for a message to travel from point (7) to point (8). The local delay is measured by the amount of time it takes for a message to travel from point (1) to point (8).

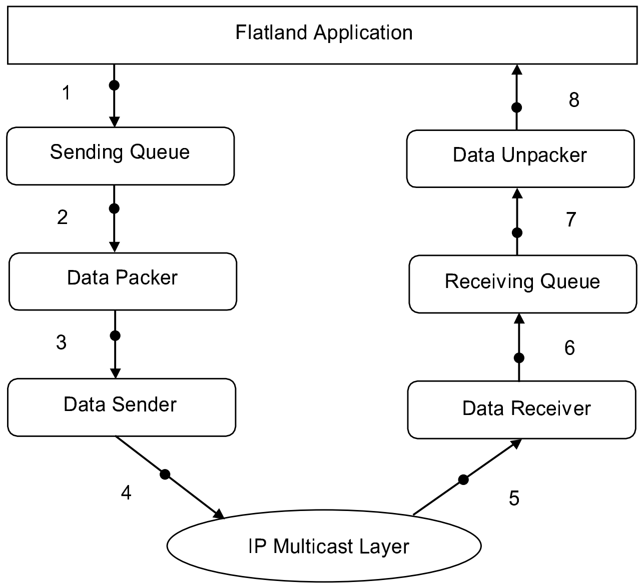


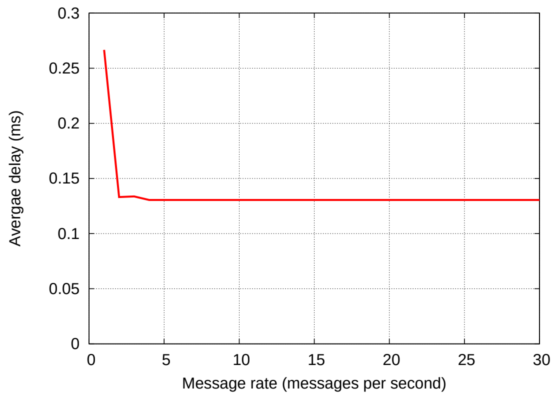
Figure 17: Message flow of a local event inside the CVE prototype system.

The simulation was configured to generate messages with different sizes at different rates. Two experiments were performed. In the first experiment, the simulation was configured to generate messages of size 150 bytes (average size of a communication message), at different rates. The experiment started by generating 1 message per second. This rate was increased by 1 message per second in each consecutive run, which lasted for 60 seconds, until 30 messages per second. The Flatland graphics frame rate was 30 frames per second. In the second experiment, the simulation was configured to generate messages with different sizes at a fixed rate (10 messages per second). A message size of 150 bytes was first used and then it was increased by 100 bytes in each consecutive run, which lasted for 60 seconds, until 8K bytes. The experiments were performed on a machine with two Pentium-III 650 MHz processors, 512 MB RAM and Linux operating system.

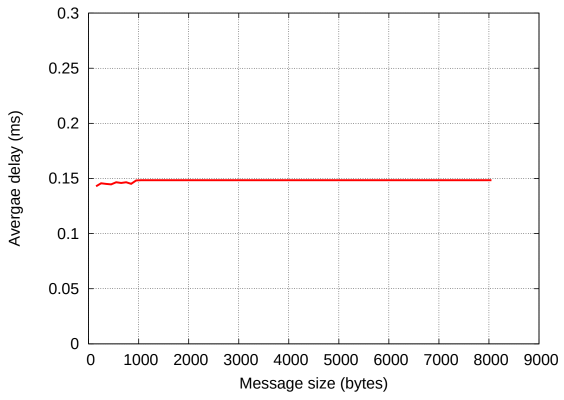
The net result of the first experiment is shown in Figure 18a. The figure shows the time average (in milliseconds) of the total local delay as a function of message generation rate (number of messages per second). The net result of the second experiment is shown in Figure 18b. The figure shows the time average (in milliseconds) of the total local delay as a function of message size (in bytes).

Internal Scalability Measurement

The following experiment was performed to measure the received message queuing delay at one node for messages generated from remote nodes in a large-scale environment. A simulation was designed to simulate the network traffic of the system in large-scale environments. Two machines were used, each with two Pentium-4 1 GHz processors, 512 MB RAM, and Linux operating system. One machine was used as a message generator (sender), while the second machine was used as a receiver. On the sender machine, messages of size 260 bytes



(a)



(b)

Figure 18: Total local delay (in milliseconds) versus (a) message rate and (b) message size.

were generated at different rates. The simulation started with 100 messages per second. This rate was increased by 100 messages per second in each consecutive run, which lasted for 60 seconds, until 10,000 messages per second. Instrumentation points (6) and (7) in Figure 17 were used to measure the received message queuing delay. Measurements were collected on the receiver machine, which runs Flatland with instrumented version of the collaboration module. The Flatland graphics frame rate was 30 frames per second. Figure 19 shows the time average (in milliseconds) of received message queuing delay as a function of message rate (number of messages per second).

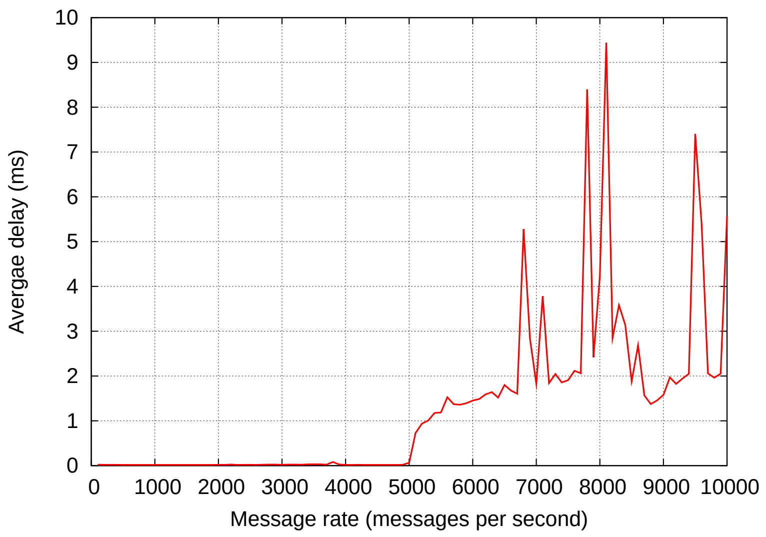


Figure 19: Queuing delay (in milliseconds) of received messages versus message rate.

3.2 External Networked Experiments

This subsection presents the results of networked experiments performed with our system. A Flatland collaborative application developed for a Telehealth project called TOUCH was used in the experiments. In the following, we first describe the collaborative application and its use in Project TOUCH, and describe the data collected during the experiments. Then, we present the results of performance measurements obtained from collected data. Finally, we present simulation results for scalability evaluation based on collected data.

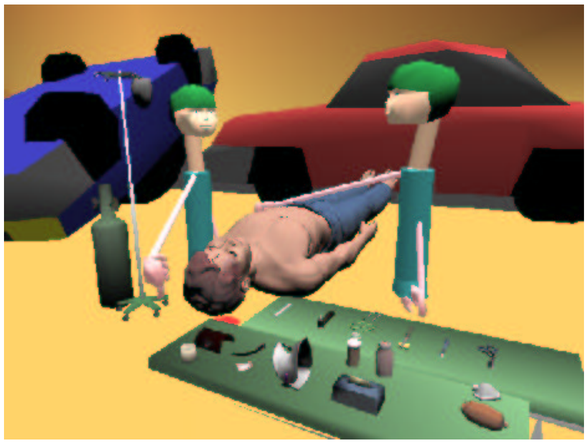


Figure 20: A view from collaborative TOUCH application.

Collaborative TOUCH Application

In the following an overview of Project TOUCH is presented, followed by description of a Flatland collaborative application used by Project TOUCH.

Project TOUCH. Project TOUCH (Telehealth Outreach for Unified Community Health) is a collaborative effort between the University of New Mexico and University of Hawaii. The purpose of the project is to demonstrate the feasibility of using advanced computing methods, such as virtual reality, to enhance education in a problem-based learning (PBL) format currently being used in the curriculum in the medicine school in both universities. The underlying technical components of Project TOUCH include Flatland tool, an artificial intelligence (AI) simulation engine, the National Computational Science Alliance's Access Grid for communication, and the software that connects them all. Project TOUCH relies on these components to enhance distributed collaboration in both first person (cohabitation of a shared virtual environment) and third person (virtual remote cameras that broadcast to remote audiences) modes of communication. The original TOUCH application, for third-person collaboration, is composed of three Flatland modules: 1) Virtual patient environment; 2) AI simulator; and 3) SpyCam virtual camera. An immersed student using joysticks and virtual controls can interact with computer-generated learning scenarios that respond logically to user interaction. Virtual patients can be simulated in any of several circumstances, with any imaginable disease or injury. The student may pick up and place Objects on the virtual patient. The activities of a participant can be monitored by faculty and other students for educational and instructional purposes.

Collaborative Application. For first-person collaboration, we used our system to enable sharing of TOUCH application. A collaboration module was added to the existing modules of TOUCH application. The collaborative TOUCH application is composed of three components: 1) Collaboration module; 2) Distributed AI Simulation; and 3) Virtual reality environment with human avatars. See Figure 20. The collaborative TOUCH application implements the main collaborative services that our system provides. Ownership management was used to control users interaction with Objects that can be picked. A user can pick an Object at any time. When a user picks an Object, he owns the Object during the time the Object is being held. The ownership of the picked Object is released when the user releases the Object.

Distributed simulations were handled as follows. Each AI Simulation module has a Flatland Object (AI Object) that is created when the module is loaded. The AI Object is replicated on all participating nodes. Among the copies of the AI Object we distinguish a master copy, which belongs to the node that loads the AI module. Other copies of the AI Object are considered as slaves. The actual simulation is located within the master copy whereas the slave copies just reflect the state of the master Object. Communication between master and slave copies is handled through INTER_OBJ messages. The master Object sends a message to itself, which is consequently distributed to the slave Objects.

The collaborative TOUCH application was used in Project TOUCH experimental sessions. During these sessions performance measurements were collected. A description of Project TOUCH experiments is presented next, followed by the results of performance measurements.

Project TOUCH Experiments. The University of New Mexico Health Sciences Center (UNMHSC) and University of Hawaii John A. Burns School of Medicine (JABSOM) have conducted several experimental sessions with Project TOUCH. During first-person collaboration sessions, the collaborative TOUCH application was used. In each session two medical students have participated. In local sessions the two students were located at the same site (UNMHSC or JABSOM), while in remote sessions one student was located at UNMHSC and the other at JABSOM. During the session, two collaborative TOUCH applications were running on two individual machines. Students were immersed in the Flatland collaborative environment and were able to interact with each other through their avatars and manipulate the virtual patient through joysticks. A total of seven sessions with fourteen students were conducted: Three remote sessions between UNMHSC and JABSOM; and four local sessions, two of them at UNMHSC and the other two at JABSOM. The average session time was 2 hours and 40 minutes. During one session, the average time during which the students were immersed was 50 minutes.

During the sessions, anonymous performance data was collected. A description of collected data and performance results are presented next.

Networked External Performance Measurement

The data collected during Project TOUCH sessions was used to measure the performance parameters defined in Section 2, namely: throughput, end-to-end delay, ownership transfer delay, response time and user-to-user time.

During each session, three machines were used: Two machines to run Flatland with the collaborative TOUCH application, and one machine at The University of New Mexico High Performance Computing Center (UNMHPC) to run the Lock Manager. The Lock Manager machine was dual-processor Pentium-III 650 MHz with 512 MB RAM and running Linux operating system. The other two machines were dual-processor Pentium-4 1 GHz with 512 MB RAM and running Linux operating system.

On each machine running Flatland, a log file was created to store the data collected during a session. The data collected in a log file contains the network traffic generated by one user during one session. No user information was logged. A total of fourteen log files were recorded during the seven sessions, two log files per session. One log file was excluded due to participant withdrawal from the immersed environment before the session was completed. Each record in a log file contains measurements of sending time, receiving time, and size of a received message, in addition to the message type and content. The collected measurements were used to obtain the throughput of received messages. The message content was used to analyze the network traffic of a specific type of messages. The messages recorded during the sessions were classified into four main types:

- MPO messages update the user's avatar position and orientation.
- PICK/DROP messages (customized INTER_OBJ messages) contain information about the Object being picked or dropped by the user.
- AI messages (customized INTER_OBJ messages) contain simulation data sent by the AI Object.
- CTRL messages contain commands control the AI simulation clock time.

The network traffic from each machine running Flatland contains messages generated from user movement (MPO messages) and user interaction (PICK/DROP messages). User movement includes head and hand movement, in addition to navigation. User interaction includes picking and dropping Objects and handing Objects to other user. Since the AI simulation was running only on one (master) machine of the two collaborating machines, the traffic from the master machine contains more messages generated from running the AI simulation (AI messages) and controlling the simulation clock time (CTRL messages). AI simulation traffic is generated according to a time-line. The AI message generation is determined according to the direction of the case scenario modified by the actions of the immersed student. The rate of AI message generation is also controlled by the operators. Seven users used the machine running the AI simulation and six users used the second machine. Traffic of users 1 through 7 contains MPO, PICK/DROP, AI, and CTRL messages while traffic of users 8 through 13 contains only MPO and PICK/DROP messages.

Figures 21a and 21b show throughput (in bits per second) of received messages from two users simultaneously immersed in the shared environment. Figure 21a shows the traffic generated by the first user (bursts) in addition to the traffic generated by the AI simulation, which was running on the first user's machine. Figure 21b shows the traffic generated by the second user. The x-axis shows the total elapsed time of the session (in seconds) including the time students were not immersed.

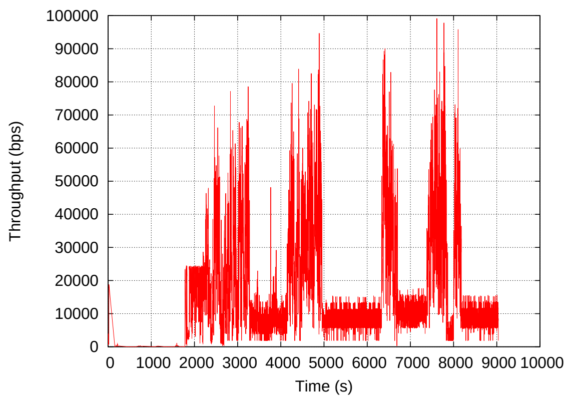
To obtain statistical measurements for the throughput of received messages, we removed the time gaps in the data stream during which students were not immersed and recalculate the throughput of the contiguous data stream. Table II shows statistics of throughput for each user. The histograms in Figure 22 show the distribution of throughput of total messages (Figure 22a), MPO messages (Figure 22b), AI messages (Figure 22c), and PICK/DROP messages (Figure 22d). The number of CTRL messages was relatively small.

The network latency between the machines used during local and remote sessions was measured using the Unix `ping` command. The `ping` command measures the round-trip time in milliseconds between two machines. Table III shows average network latency (round-trip time divided by two) between two collaborating machines, and average network latency between one collaborating machine and the machine running Lock Manager.

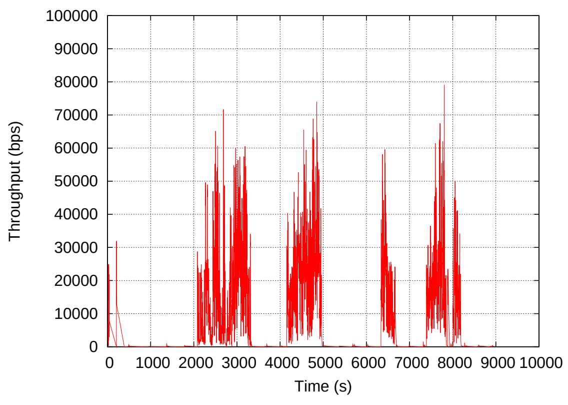
The performance parameters end-to-end delay, ownership transfer delay, response time and user-to-user time were obtained using the measured network latency and local delay measured in the local experiments. The end-to-end delay is the sum of network latency and local delay. The ownership transfer delay is one round-trip time between the machine running Flatland and the machine running Lock Manager. The response time is the sum of ownership transfer delay and local delay. The user-to-user time is the sum of ownership transfer delay and end-to-end delay. Table IV shows average end-to-end delay between two users during local and remote sessions. Table V shows average ownership transfer delay, response time and user-to-user time for two users during local and remote sessions.

User	Mean	Median	Std Dev	Minimum	Maximum
User 1	28438	26904	15771	173	93230
User 2	25870	22843	16316	104	92460
User 3	25686	24128	14509	252	92781
User 4	32672	31974	17167	254	99174
User 5	28968	28627	15204	202	93124
User 6	32709	30592	18078	1083	101358
User 7	30024	27561	18370	333	105398
User 8	23157	21470	14818	46	90683
User 9	19737	18550	12865	168	66068
User 10	21455	19490	14505	50	83957
User 11	20764	20081	13396	95	79077
User 12	20511	19745	12891	103	67218
User 13	22187	21805	12890	417	65748
Average	25552	24136	15137	252	86944

Table II: Statistics of throughput (bits per seconds) for the users participating in Project TOUCH sessions.

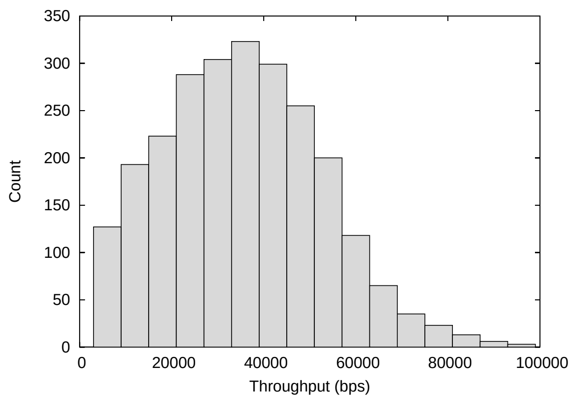


(a)

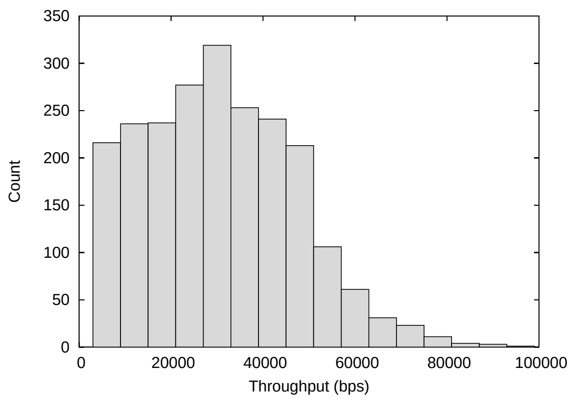


(b)

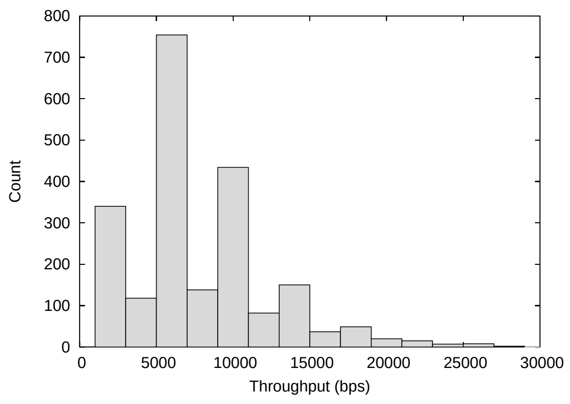
Figure 21: Throughput (bits per seconds) for (a) user 1 with AI simulation and (b) user 2.



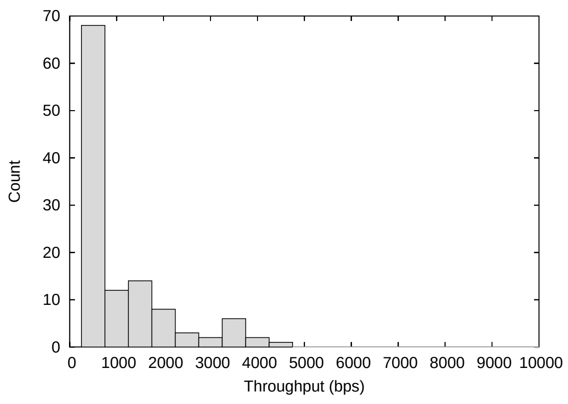
(a)



(b)



(c)



(d)

Figure 22: Histogram of throughput (bits per seconds) of (a) total messages, (b) MPO messages, (c) AI messages, and (d) PICK/DROP messages for one user.

Sites	Machines	Network Latency
UNMHSC - JABSOM	flatland1.hsc.unm.edu - flatland2.biomed.hawaii.edu	53.80 *ms*
UNMHSC	flatland1.hsc.unm.edu - flatland2.hsc.unm.edu	0.22 *ms*
JABSOM	flatland1.biomed.hawaii.edu - flatland2.biomed.hawaii.edu	0.18 *ms*
UNMHSC - UNMHPC	flatland1.hsc.unm.edu - lockmanager.hpc.unm.edu	1.78 *ms*
	flatland2.hsc.unm.edu - lockmanager.hpc.unm.edu	1.72 *ms*
JABSOM - UNMHPC	flatland1.biomed.hawaii.edu - lockmanager.hpc.unm.edu	50.19 *ms*
	flatland2.biomed.hawaii.edu - lockmanager.hpc.unm.edu	50.09 *ms*

Table III: Average network latency (in milliseconds) between the machines used during Project TOUCH sessions.

Session	End-to-End Delay
Remote (UNMHSC - JABSOM)	53.93 *ms*
Local (UNMHSC)	0.36 *ms*
Local (JABSOM)	0.32 *ms*

Table IV: Average end-to-end delay (in milliseconds) during Project TOUCH sessions.

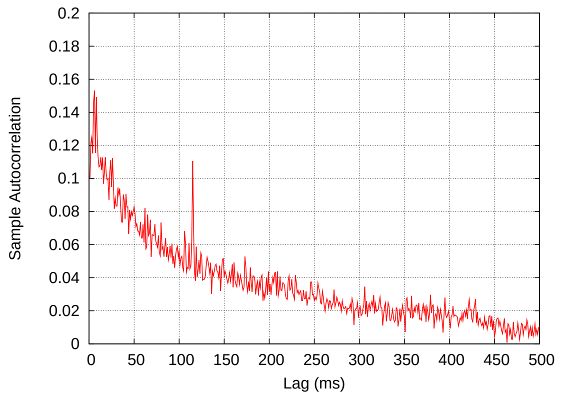
Session	User	Ownership Transfer Delay	Response Time	User-to-User Time
Remote	User 1 (UNMHSC)	3.56 *ms*	3.70 *ms*	154.31 *ms*
	User 2 (JABSOM)	100.38 *ms*	100.52 *ms*	57.49 *ms*
Local (UNMHSC)	User 1	3.56 *ms*	3.70 *ms*	3.80 *ms*
	User 2	3.44 *ms*	3.58 *ms*	3.92 *ms*
Local (JABSOM)	User 1	100.38 *ms*	100.52 *ms*	100.50 *ms*
	User 2	100.18 *ms*	100.32 *ms*	100.70 *ms*

Table V: Average ownership transfer delay, response time, and user-to-user time (in milliseconds) during Project TOUCH sessions.

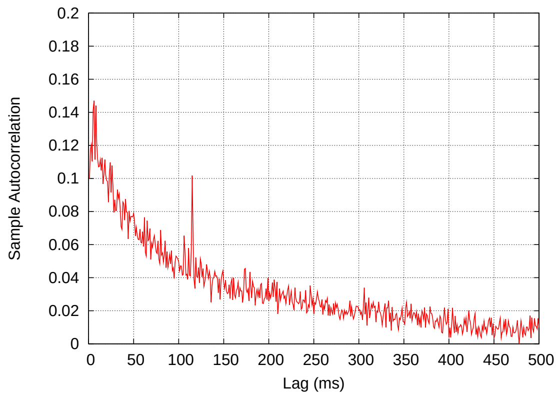
We have presented the results of performance measurements obtained from the data collected during Project TOUCH sessions. Next, we present the results of scalability measurements obtained through simulation using the data collected during the sessions.

Networked External Scalability Measurement

The data collected from Project TOUCH sessions was used to measure the scalability of our system based on use patterns during the sessions. We used the log files recorded during the sessions to generate a contiguous stream of received messages for each user who participated in the sessions. Each message stream contains interarrival times of received messages. The interarrival time is the time interval between two successive arrival messages. A total of 13 streams represent the network traffic of 13 users were generated. A simulation was designed to perform a set of experiments. The input to the simulation was the actual message streams of 13 users participated in Project TOUCH sessions and randomly shuffled versions of the actual streams [9]. Figures 23a and 23b show the autocorrelation of interarrival times of the actual message stream and a shuffled version of it for one user, respectively.



(a)



(b)

Figure 23: Autocorrelation of interarrival times for one user: (a) actual data and (b) shuffled data.

The simulation was run on six machines each with a Linux operating system: Four machines were dual-processor Pentium-4 1 GHz with 512 MB RAM; and two machines were dual-processor Pentium-III 650 MHz with 512 MB RAM. During the simulation five machines were used as a message generator (sender) while one machine was used as a receiver. The actual and shuffled message streams were used to generate the messages. Each message stream represents the network traffic of one user participated in Project TOUCH sessions. Arrival times of received messages were measured on the receiver machine, which runs Flatland with the collaboration module. The Flatland graphics frame rate was 30 frames per second.

A series of experiments were performed with an increasing number of users (message streams). In the first experiment, the actual unshuffled message streams (13 streams) were used. In the next experiments, an additional 13 shuffled streams were added in each experiment until 104 streams, simulating new users. Table VI shows statistics of the throughput of received messages for number of users in each experiment. The histograms in Figure 24 show the distribution of the throughput of messages received from 13 users (Figure 24a), 26 users (Figure 24b), 52 users (Figure 24c), and 104 users (Figure 24d). Figure 25 shows the average throughput (in bits per second) versus the number of users.

This section has presented the results of local and external networked experiments performed with our prototype system. Next section discusses and analyzes these results.

# of users	Mean	Median	Std. Dev.	Minimum	Maximum
1 user	25552	24136	15137	252	86944
13 users	301148	297461	61299	71200	515171
26 users	621218	619959	80211	319237	965567
39 users	899956	898284	101104	409989	1222590
52 users	1211078	1211361	115111	319248	1616486
65 users	1523299	1523276	150770	300493	2111088
78 users	1845021	1850613	157947	794660	2344192
91 users	2060873	2069806	179423	667124	2585393
104 users	2246443	2305410	333820	267408	3547775

Table VI: Statistics of throughput (bits per second) for different numbers of users.

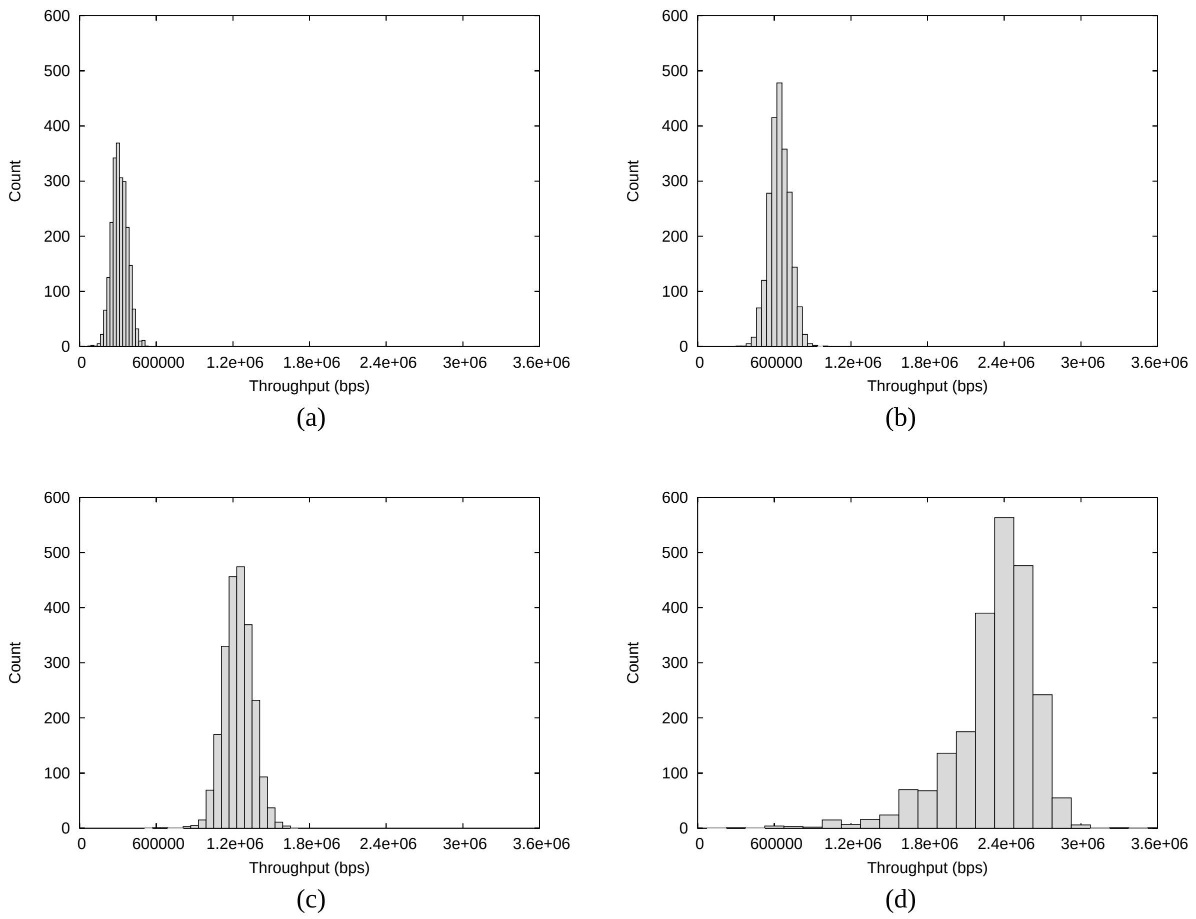


Figure 24: Histogram of throughput (bits per second) for (a) 13 users, (b) 26 users, (c) 52 users, and (d) 104 users.

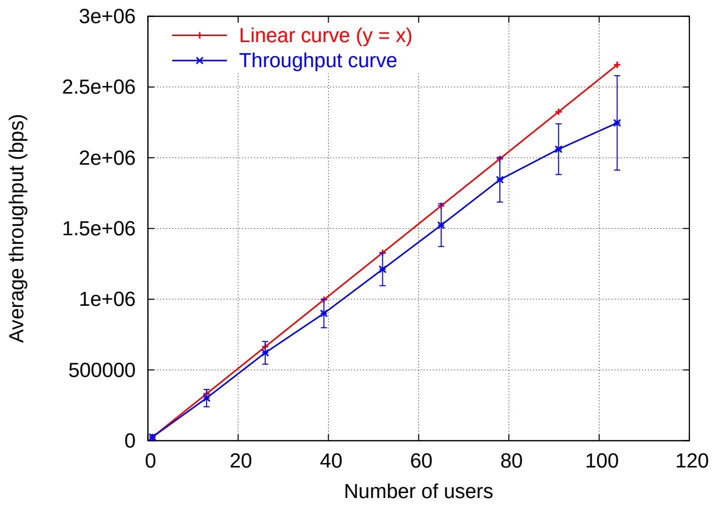


Figure 25: Simulation results of average throughput (bits per second) versus number of users.

4 Discussion

The prototype implementation of the framework described in Section 2 exposed each of the issues presented in Section 1 through the services provided by the collaboration module. Specifically, the data distribution scheme handled the scalability and network bandwidth limitation, the ownership management dealt with concurrent actions by multiple users to avoid inconsistencies, the late-join handling mechanism allowed new users to join an ongoing collaboration session and get the current state of the shared environment, and the avatar management dealt with the issues associated with user representation within the virtual environment. To evaluate the performance of the implemented prototype, we conducted two kinds of experiments: local and networked, described in Section 3. The local experiments focused on the internal system characterization, while the external networked experiments focused on the effects of specific issues on the system performance. Specifically, we evaluated the approaches to data distribution, ownership management, and scalability. This section discusses the results obtained from the local and external networked experiments.

4.1 Local Experiments

The results from the local experiments were used to characterize the internal system design. In the following, we discuss the results obtained from the experiments to evaluate the system performance and scalability.

Local Performance Evaluation

The system characterization focuses on performance evaluation of the collaboration model used by the system. In our evaluation approach described in Section 2 we defined six performance parameters to measure, namely: generated message queuing delay, generated message packing delay, loop-back delay, received message queuing delay, received message unpacking delay and local delay. These parameters determine the time between local event generation and execution. For real-time interaction, the above delays should be as small and as constant as possible.

Two local experiments were described in Section 3. The first experiment addressed the effect of increasing number of local events (update messages) generated per second on the local delay introduced by the collaboration model, using a fixed message size. The second experiment addressed the effect of increasing the size of update messages on the local delay, using a fixed message generation rate.

The results of the first experiment show that the delays introduced by the collaboration model are constant and small (for real-time interaction) for various message rates, up to 30 messages per second. Figure 18a shows that the average local delay is less than 0.14 milliseconds. The results of the second experiment show that the delays introduced by the collaboration model are constant and small (for real-time interaction) for increasing message sizes, up to 8 KB. Figure 18b shows that the average local delay is less than 0.15 milliseconds.

The above results from both experiments show that the collaboration model introduces minimal delay (a few hundred microseconds) between local event generation and execution, with respect to real-time interaction.

Local Scalability Evaluation

The scalability evaluation of the collaboration model focuses on the effect of increasing message throughput on received message queuing delay at one node in a large-scale environment. The simulation results show that the average queuing delay is less than 22 microseconds for message rate up to 5,000 messages per second (Figure 19). The results also show that the queuing delay significantly increases when message rate is larger than 5,000 messages per second. For instance, the average queuing delay was 9.5 milliseconds for message rate of 8,100 messages per second.

Increasing the message rate causes more contention on the shared queue between the thread that receives the messages and the thread that processes them. In the collaboration model, the queuing operation involves thread synchronization using mutual exclusion lock, or mutex, to control access to the shared queue. Results from performance analysis of the LinuxThreads library [15] (used in our system) showed that the time it takes for thread synchronization is between 100 to 200 microseconds, and the overhead time of mutex lock/unlock with contention increases dramatically compared to the overhead with no contention. The results in Figure 19 show that the contention between the collaboration threads becomes significant when the message rate is larger than 5,000 messages per second, or the time interval between two successive messages is less than 200 microseconds. From the above results we can conclude that the collaboration model performs well when there is no contention between the collaboration threads, and increasing this contention limits the scalability of the system.

The results of the performance and scalability evaluation obtained from the local experiments indicated that the prototype of our system provides efficient performance, and pointed out limitations in scalability of the current implementation.

4.2 External Networked Experiments

The results from the external networked experiments were used to evaluate the effects of specific issues on the system performance. In our evaluation approach described in Section 2 we considered three issues, namely: data distribution, ownership management and scalability. In the following, we discuss the results obtained from the networked experiments.

Data Distribution Evaluation

To evaluate the data distribution scheme used by our system, we measured two performance parameters: throughput and end-to-end delay. In the following, we discuss the results obtained for each parameter.

Figures 21a and 21b show throughput versus total elapsed time for two users simultaneously immersed in the shared environment, with the AI simulation running on the first user's machine (Figure 21a). In Figure 21a, the time periods with low throughput (less than 20 Kbps) represent the time during which the simulation was running and no users were immersed. The time periods with minimal throughput represent the time during which no users were immersed and no simulation was running. This minimal throughput is due to the positional noise caused by interfering fields with the electromagnetic trackers for user's head and hand movement. In Figure 21b, the time periods with minimal throughput represent the time during which no users were immersed. Since the AI simulation was running only on the machine that was used by the first user, no simulation traffic appears in this figure.

The histograms for one user throughput of total messages (Figure 22a), MPO messages (Figure 22b), AI messages (Figure 22c), and PICK/DROP messages (Figure 22d) show that the received messages from one user are mainly MPO messages. This indicates that most of the user traffic is generated from user motion and movement. The average throughput of total received messages from each user in Table II represents the average consumed bandwidth per user. The statistics in the table show that the average user bandwidth is 25 Kbps with standard deviation of 15 Kbps. This low bandwidth requirement allows the system users to run collaborative applications (with user activities similar to those in TOUCH sessions) over relatively low-bandwidth networks.

The second performance parameter measured for data distribution evaluation is end-to-end delay. The average end-to-end delay in Table IV is measured as the sum of network latency and local delay. The results for remote sessions show the expected result that end-to-end delay is highly affected by network latency. For example, during remote sessions, average network latency was 53.80 milliseconds (Table III) and average end-to-end delay was 53.93 milliseconds (Table IV). For local sessions the network latency was small and close to local delay (0.18 - 0.22 milliseconds, Table III).

Ownership Management Evaluation

To evaluate the ownership management scheme used by our system, we measured three performance parameters: ownership transfer delay, response time, and user-to-user time. In the following, we discuss the results obtained for each parameter.

The ownership transfer delay in Table V is measured as the round-trip time between the machine running Flatland (at UNMHSC or JABSOM) and the machine running Lock Manager (at UNMHPC). The results show that ownership transfer delay depends on the connection between Lock Manager and the node requesting ownership. For example, users at UNMHSC experienced less delay than those at JABSOM since the Lock Manager machine was at UNMHPC. Both users during a local session experienced the same ownership transfer delay (On average, 3.5 milliseconds during UNMHSC local sessions, and 100.28 milliseconds during JABSOM local sessions). During a remote session, each user experienced different ownership transfer delay. User at UNMHSC experienced average delay of 3.56 milliseconds while user at JABSOM experienced average delay of 100.38 milliseconds.

The response time in Table V is measured as the sum of ownership transfer delay and local delay. The results for both local and remote sessions show that response time is highly affected by ownership transfer delay. For example, the average response time for users at UNMHSC is 3.66 milliseconds and the average ownership transfer delay is 3.52 milliseconds. For users at JABSOM, the average response time is 100.45 milliseconds and the average ownership transfer delay is 100.31 milliseconds.

The user-to-user time in Table V is measured as the sum of ownership transfer delay and end-to-end delay. The results show that user-to-user time is affected by both ownership transfer delay and end-to-end delay. For example, user at UNMHSC sees the action of user at JABSOM after 154.31 milliseconds. This time includes the time it takes for JABSOM user to acquire ownership (100.38 milliseconds) and send update message to UNMHSC user (53.93 milliseconds).

Networked External Scalability Evaluation

The scalability evaluation through local experiments, discussed in the previous subsection, focused on the collaboration model and the effect of increasing message throughput on received message queuing delay, using uniform use patterns. The goal of the scalability evaluation through external networked experiments is to evaluate the system scalability under real-world scenarios. This evaluation examines the scaling of throughput with number of users having different use patterns.

In the simulation experiments, we used the actual message streams of 13 users and certain randomly shuffled versions of the actual streams. The idea behind using shuffled versions of the actual stream is to obtain message streams with identical message interarrival time distributions but with different autocorrelation between interarrival times. In our experiments we considered shuffling that eliminates long-range correlations, while preserves short-range correlations. Figures 23a and 23b show very similar autocorrelation functions up to lag of 500 milliseconds for interarrival times of the actual message stream and a shuffled version of it for one user, respectively. The histograms of throughput in Figure 24 for the number of users used in each experiment show similar throughput distributions but with increasing mean value. The graph of throughput versus number of users in Figure 25 shows the experimental and theoretical curves of throughput scalability. The error bars in the graph represent the standard deviation of the measured throughput (See Table VI). The results show that the throughput scales sublinearly with the number of users. This scalability curve can help to determine the expected performance of the system under real-world scenarios more accurately than using the theoretical curve (linear curve with a slope of 1), which overestimates the bandwidth requirements.

The experimental results showed that the prototype provides efficient performance, for collaborative applications with requirements similar to those presented in TOUCH application. The flexibility and openness of the implemented prototype allow us to experiment with other approaches that may further optimize the system performance, and integrate new techniques to support various collaborative applications. Future work for further optimization and evaluation is discussed in the next section.

5 Conclusions and Future Work

In this document we presented a framework for developing flexible and extensible Collaborative Virtual Environments. The framework comprehensively addresses all of the major issues associated with CVE design and implementation. A flexible open prototype system based on this framework was developed. The prototype explicitly exposed each of the design issues presented in Section 1. It used particular approaches as instances to deal with each issue. The overall system performance was systematically evaluated through internal focus on internal system characterization and external focus on the effects of specific issues on the system performance. The experimental results showed that the implemented prototype provides efficient performance for collaborative applications with requirements similar to those presented in TOUCH application. The evaluation of our prototype implementation shows that our approach is feasible, and can be applied for the evaluation of other instances of issues.

The flexibility and extensibility of the implemented prototype based on the proposed framework open the way for further research in the area of CVEs. Future work includes:

- Investigation of new and alternative techniques and approaches. For example, using optimistic approaches for concurrency control instead of the current pessimistic mechanism, or developing a hybrid concurrency control that supports different application requirements. For ownership management in pessimistic concurrency control, distributed approaches can be used instead of the current centralized approach. In network communication, different communication protocols and delivery mechanisms for different data types can be used for efficient data distribution.
- Incorporation of new technologies such as haptics [3] and large displays (powerwalls) [21]. Haptics is a recent enhancement to virtual environments allowing users to touch and feel the virtual objects they interact with. Current haptic devices allow force feedback from manipulated objects through haptic interfaces (such as the PHANToM^{tm} arm [30]). Haptics in CVEs allows users to feel a shared object as well as to feel each others' actions, permitting more natural interaction. The force control algorithms used for haptic rendering generally require high sampling rates and low latency [4]. Future work will explore the haptic issues in CVEs and adding this technology to our system. Another technology used in virtual reality is the large displays in which a large display is divided into several tiles, each of which is managed by a corresponding computer (node). These displays are usually used to project a large, high-resolution visualization. Multi-display visualization based on distributed rendering introduces two issues: synchronization of distributed visualization and handling distributed interaction [10]. Future work will exploit the framework to provide support for multi-display visualization controlled by multiple users.
- Supporting parallel and distributed simulations. In the current implementation we use a centralized approach to support distributed simulations. In this approach, a dedicated single node runs the simulation and distributes the results to other nodes for visualization. To avoid centralization and improve system scalability, we plan to adopt a distributed approach where the simulation is distributed among multiple nodes [13]. A key problem in distributed approaches is to keep the simulation synchronized among multiple nodes. Time synchronization in distributed simulations depends on the time advance mechanism. For example, in continuous time simulations, state changes occur continuously across time, while in discrete time simulations, state changes only occur at discrete time instants. Future work will investigate the time synchronization issues and the current synchronization techniques for large-scale distributed simulations.

References

[1] J. W. Barrus, R. C. Waters, and D. B. Anderson. Locales and beacons: Efficient and precise support for large multi-user virtual environments. In *Proceedings of IEEE Virtual Reality Annual International Symposium (VRAIS '96)*, pages 204–213,, 1996.

[2] S. Benford, J. Bowers, L.E. Fahlen, C. Greenhalgh, J. Mariani, and T. Rodden. Networked virtual reality and cooperative work. *Presence: Teleoperators and Virtual Environments*, 4(4):364–386, 1995.

[3] G. Burdea. *Force and Touch Feedback for Virtual Reality*. John Wiley and Sons, New York, 1996.

[4] G. Burdea. Haptic issues in virtual environments. In *Proceeding of Computer International 2000*, Geneva , Switzerland, June 2000.

[5] T. Capin, H. Noser, D. Thalmann, I. Pandzic, and N. Thalmann. Virtual human representation and communication in VLNET. *IEEE Computer Graphics and Applications*, 17(2):42–53, March-April 1997.

[6] T. Capin, I. Pandzic, N. Magnenat Thalmann, and D. Thalmann. *Avatars in Networked Virtual Environments*. John Wiley & Sons, 1999.

[7] M. Capps, D. McGregor, D. Brutzman, and M. Zyda. NPSNET-V: A new beginning for dynamically extensible virtual environments. *IEEE Computer Graphics and Application*, pages 12–15, September-October 2000.

[8] Standard for information technology - Protocols for distributed interactive simulation applications. Technical Report IST-CR-93-15, Institute for Simulation and Training, May 1993.

[9] A. Erramilli, O. Narayan, and W. Willinger. Experimental queueing analysis with long-range dependent packet traffic. *IEEE/ACM Transactions on Networking*, 4(2), 1996.

[10] J. M. Frahm, J. Evers-Senne, and R. Koch. Network protocol for interaction and scalable distributed visualization. In *1st International Symposium on 3D Data Processing Visualization Transmission*, Padova, Italy, June 2002.

[11] E. Frecon and M. Stenius. DIVE: A scalable network architecture for distributed virtual environments. *Distributed Systems Engineering Journal*, 5(3):91–100, 1998.

[12] L. Freitag, D. Diachin, D. Heath, J. Herzog, and P. Plassmann. Remote engineering using cave-to-cave communications. In *Virtual Environments and Distributed Computing at Supercomputing '95: GII Testbed and HPC Challenge Applications on the I-Way*, volume 41, 1995.

[13] R. Fujimoto. Parallel and distributed simulation. In *Winter Simulation Conference*, pages 122–131, 1999.

[14] T. A. Funkhouser. Network topologies for scalable multi-user virtual environments. In *Proceedings of the Virtual Reality Annual International Symposium (VRAIS '96)*, pages 222–228, 1996.

[15] F. Garcia and J. Fernandez. POSIX thread libraries. *Linux Journal*, (70), 2000. http://www.linuxjournal.com.

[16] S. Greenberg and D. Marwood. Real time groupware as a distributed system: Concurrency control and its effect on the interface. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (ACM CSCW '94)*, pages 207–217, October 1994.

[17] C. Greenhalgh and S. Benford. MASSIVE: A collaborative virtual environment for teleconferencing. *ACM Transactions on Computer-Human Interaction*, 1995.

[18] C. Greenhalgh and S. Benford. MASSIVE: A distributed virtual reality system incorporating spatial trading. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, pages 27–34, 1995.

[19] C. Greenhalgh, J. Purbrick, and D. Snowdon. Inside MASSIVE-3: Flexible support for data consistency and world structuring. In *Proceedings of ACM Conference on Collaborative Virtual Environments (CVE 2000)*, pages 119–127, September 2000.

[20] C. Greenhalgh and I. Vaghi. Consistency in shared virtual environments. Technical report, University of Nottingham, 1998.

[21] G. Humphreys and P. Hanrahan. A distributed graphics system for large tiled displays. In *Proceedings of IEEE Visualization Conference*, pages 215–223, 1999.

[22] A. Kapolka, D. McGregor, and M. Capps. A unified component framework for dynamically extensible virtual environments. In *Proceedings of the Fourth ACM International Conference on Collaborative Virtual Environments*, September 2002.

[23] N. Kuijpers, P. V. Gool, and H. Jense. A component architecture for simulator development. In *Proceedings of 1998 Spring Simulation Interoperability Workshop*, March 1998.

[24] V. Kumar. *MBone: Interactive Multimedia on the Internet*. New Riders, 1996.

[25] R. Lea, Y. Honda, K. Matsuda, O. Hagsand, and M. Stenius. Issues in the design of a scalable shared virtual environment for the Internet. In *30th Hawaii International Conference on System Sciences (HICSS '97*, volume 1, 1997.

[26] J. Leigh and A. E. Johnson. Supporting transcontinental collaborative work in persistent virtual environments. *IEEE Computer Graphics and Applications*, pages 47–51, 1996.

[27] J. C. Lin and S. Paul. RMTP: A reliable multicast transport protocol. In *Proceedings of IEEE INFOCOM '96*, pages 1414–1424, 1996.

[28] M. Macedonia and M. Zyda. A taxonomy for networked virtual environments. In *Proceedings of the 1995 Workshop on Networked Realities*, October 1995.

[29] M. Macedonia, M. Zyda, D. Pratt, P. Barham, and S. Zestwitz. NPSNET: A network software architecture for large-scale virtual environments. *Presence: Teleoperators and Virtual Environments*, 3(4), 1994.

[30] T. Massie and K. Salisbury. The PHANToM haptic interface: A device for probing virtual objects. In *Proceedings of the ASME Dynamic Systems and Control Division*, volume 55-1, pages 295–300, 1994.

[31] T. Maufer. *Deploying IP Multicast in the Enterprise*. Prentice Hall, 1998.

[32] D. Miller. Long-haul networking of simulators. In *Proceedings of Tenth Interservice/Industry Training Systems Conference*, 1989.

[33] G. Singh, L. Serra, W. Png, and H. Ng. BrickNet: A software toolkit for network-based virtual worlds. *Presence: Teleoperators and Virtual Environments*, 3(1):19–34, 1994.

[34] S. Singhal and D. Cheriton. Using projection aggregations to support scalability in distributed simulation. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS '96)*, 1996.

[35] D. Snowdon and A. West. AVIARY: Design issues for future large-scale virtual environments. *Presence: Teleoperators and Virtual Environments*, 3(4):288–308, 1994.

[36] U. Sung, J. Yang, and K. Wohn. Concurrency control in CIAO. In *IEEE Virtual Reality '99*, pages 22–28, 1999.

[37] R.C. Waters, D.B. Anderson, and D.L. Schwenke. Design of the interactive sharing transfer protocol. In *IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (IEEE WETICE '94)*, pages 140–147, 1994.

[38] K. Watsen and M. Zyda. Bamboo: A portable system for dynamically extensible, real-time, networked, virtual environments. In *Proceedings of IEEE Virtual Reality Annual International Symposium (VRAIS '98)*, March 1998.