Managing Collaboration in the nanoManipulator

Thomas C. Hudson
University of North Carolina at Wilmington
hudsont@uncw.edu

Aron T. Helser
3rdTech, Inc.
helser@3rdtech.com

Diane H. Sonnenwald, Mary C. Whitton
University of North Carolina at Chapel Hill
dhs@ils.unc.edu, whitton@cs.unc.edu

Abstract

We designed, developed, deployed, and evaluated the Collaborative nanoManipulator (CnM), a system supporting remote collaboration between users of the nanoManipulator interface to atomic force microscopes. To be accepted by users, the shared nanoManipulator application had to have the same high level of interactivity as the single user system and the application had to support a user's ability to interleave working privately and working collaboratively. This paper briefly describes the entire collaboration system, but focuses on the shared nanoManipulator application. Based on our experience developing the CnM, we present: a method of analyzing applications to characterize the requirements for sharing data between collaborating sites, examples of data structures that support collaboration, and guidelines for selecting appropriate synchronization and concurrency control schemes.

Keywords: state replication, collaborative virtual environments, scientific collaboration, distributed collaboration, concurrency control, nanoManipulator

1. Introduction

A large number of virtual environment (VE) applications can be categorized as involving one or more users moving around in and exploring a world model. Examples include visualization and analysis of scientific or other abstract data, design reviews, and shared, purely experiential VEs for education or entertainment. While in this type of VE, users may vary their viewpoints, vary the viewing parameters, use analysis tools, and generate annotations and other supplemental data.

In these exploratory VEs, the users seldom modify the world model during the VE session. However, as with live data from an atomic force microscope (AFM), the model need not be static. It may come from a real-time sensor, be generated by a simulation, or be the playback of a recorded sequence from one of those sources.

In a multi-year, NIH-funded project designed to discover whether collaborators working remotely can perform "good" science, we designed, developed, deployed, and evaluated a distributed, collaborative version of the nanoManipulator (CnM) system developed at UNC Chapel Hill [1, 2]. The CnM system is in use and has been the subject of both a controlled experimental evaluation and an ongoing ethnographic evaluation. This paper briefly describes the entire collaboration system including the hardware and software for the nM application as well as for video conferencing and shared productivity applications. The paper, however, focuses on the design of the shared nM application.

Two requirements drove the architecture of the shared nanoManipulator application: it had to be highly interactive, with both response time and user-to-user time less than 300 ms, and it had to support the ability for a collaborator to interleave periods of working independently (private mode) with periods of working collaboratively (shared mode), with the associated requirement that there be an easy mechanism for copying application state data back and forth between the independent (private) and collaborative (shared) modes.

The interactivity requirement drove the early design decision to implement the collaborative system with replicated copies of the application at each site. Once that decision was made, we then addressed a) the problem of characterizing the control parameters in our system so that we could match them to appropriate synchronization and concurrency control mechanisms and b) the problem of creating and managing the two sets of state data required at each site to support both shared and private work. We found that an extension of the model-view-controller paradigm [3] improved our understanding of the system, leading to a better architecture and improved maintainability.

In this paper we present an extension of the model-view-controller paradigm to help developers of collaborative VE systems analyze applications and characterize control parameters. We also present a set of guidelines for selecting an appropriate concurrency control technique for varying types of system data.

2. Background

2.1. Interactivity

In addition to specific functionality, chief among any set of system requirements is usability, and latency is a significant determinant of the usability of an interface. Graphical and VE applications have particularly stringent latency requirements because their interfaces are *interactive*: users directly manipulate parameters controlling the images they see using continuous input devices such as mice. The usability of interactive interfaces degrades significantly when visual feedback is not essentially immediate.

Collaborative applications must address two distinct types of latency: response time and user-to-user time [4]. *Response Time* is the time between a user's input and that user seeing the results of his input. *User-to-User Time* is the time between a user's input and a remote collaborator seeing the results of that input. Different aspects of the system design contribute to these two latencies; both must be minimized to create a high-performance VE system.

Response times must be tightly bounded in interactive interfaces. The VE community has reported 100 ms as an approximate upper bound for direct manipulation tasks [5-7]. In the context of a two-dimensional collaborative application, Bhola *et al.* state that the limit varies between 50 and 100 ms [4]. When force feedback is being used simultaneously, or user input is driving a control loop, response time becomes even more critical: 50 ms of latency in flight simulators reduces performance, and only a little more is needed to cause system instability [8].

When users are attempting to coordinate their actions, or understand how their actions affect one another, user-to-user time is important. Several authors report 200 ms of delay interfering with the completion of closely coordinated tasks in collaborative virtual environments [9].

2.2. Concurrency Control

Concurrency control mechanisms attempt to guarantee that all users see a consistent (i.e., the same) application state at the same time. If users expect to see the same thing and discover that they do not, the collaboration will break down. The latency requirements of interactive graphical applications make concurrency control a major issue in effectively data sharing between geographically separated users.

Concurrency control mechanisms also attempt to guarantee that users always see an expected application state, sometimes expressed as the *Principle of No Surprises*, or the *Isolation Property* of application state data. Users should never be surprised by the application's response to their input or see unintended effects due to a collaborator's simultaneous actions.

A third requirement for concurrency control mechanisms is that they prevent application behavior that may cause harm to data or devices. "A concurrency-control protocol ensures that incorrect behavior cannot occur as a result of concurrent access by multiple clients." (Herlihy, pg 249) [10].

Achieving Concurrency—Locks. If an application is centralized and uses locks for concurrency control, all clients suffer delay of one network round-trip time on every operation. To speed up client operations, state can be replicated at every client's computer. Clients can then quickly read local state, but need to execute a distributed concurrency control protocol to obtain and release locks when writing to their own copy of the application's state and to disseminate the results of writes to other users.

Floor control is the simplest application of database-style locking to collaborative applications. Floor control uses a single lock to guard access to the entire application or document being shared [11]. Only one user at a time can actively manipulate the application. Inactive users press a button to request control of the application. As in a well-run meeting, only one speaker at a time "has the floor" and can address others or take action, while others watch and listen. Microsoft's NetMeeting™ is one familiar product that uses floor control.

Locks can be made "fine-grained". For example, a document can be divided up into many pieces with one lock controlling each piece; only one user can manipulate a given section of a document, but users may work in parallel on different sections of the document. This increases the need to interleave lock management with the user's natural workflow, breaking concentration on the task and usually giving the user a significantly worse experience than the single-user application. Fine-grained locks can be implicit: rather than explicitly requesting a lock, the user begins a manipulation; the system then checks to see if that part of the document is locked, and if so aborts their operation. Implicit locks avoid the workflow disruption of explicit locking, but increase application latency, and can cause an unexpected loss of work when operations are aborted, violating the "No surprises" directive and greatly discouraging users.

Achieving Concurrency—Optimism. Optimistic concurrency control is a class of algorithms that guarantees consistency with minimal latency. Following Amdahl's Law — *make the common case fast* — systems using optimistic concurrency control perform an operation, then check to see whether the operation conflicted with other simultaneous operations at different hosts. If conflicts occur, an application-dependent procedure resolves them [12]. Since most operations do not cause conflicts, they can be executed rapidly, without the time penalty of locking. Many groupware systems use optimistic concurrency control because of their sensitivity to latency,

Optimistic concurrency control has to be careful to preserve the intention of the multiple concurrent users of the system. Recent work has extended optimistic approaches from text editors [13] to spreadsheets and 2D graphics editors. The work reported in this paper further extends them to an interactive 3D graphics application.

2.3. Collaborative Virtual Environments

Meehan [14] surveyed multi-user distributed virtual-environment (DVE) systems, showing the wide range of concurrency control approaches that have been tried. The NPSNET and AVIARY systems are of particular interest because they avoid the latency penalty of locking protocols [15, 16].

Concurrency control has not been the focus of prior massively multi-user DVE systems, for which bandwidth is a more serious constraint than network latency. However, in the nanoManipulator with its relatively small amounts of data to be transferred, the network's inherent latency is a much larger component of user-to-user latency than is bandwidth: propagation and queuing times are much greater than transmission time; distance and network congestion contribute more to user-to-user latency than does bandwidth.

In this work, we assumed that network latency is beyond the application's control. Instead of trying to control network performance, we change the implementation of the application to reduce its demands of the network – to improve application-level performance for a given network performance. Another tactic is to modify the application's user interface, to help users adapt to latency. A good survey of this approach, one not used in our system, can be found in [9].

Our research was performed in the context of replicated collaborative virtual environments (CVEs) with small numbers of users. Replication scales poorly; systems with hundreds and thousands of users require more complex concurrency support than used in our system, typically including area-of-interest management and the reintroduction of servers to filter communication [16]. However, these added layers of distributed architecture do not invalidate our analysis and implementation techniques.

2.4. Collaborative Science

Scientific collaboratories are a hot topic today.; our project is one of eight funded by the NIH to evaluate collaboration technology. Notable work has been done at the Beckman Institute where researchers have created a number of programs that allow the sharing of a Scanning Electron Microscope (SEM), including Bugscope [17], a tool for K-12 teachers. Another scientific collaboratory, the Upper Atmospheric Research Collaboratory (UARC), provides shared access to a number of scientific instruments that observe Earth's ionosphere [18]. The Beckman system is typical in that users have a web-based interface that shows them one frame of video captured by the SEM. Buttons on this interface cause the SEM to pan, zoom, or otherwise modify the SEM control parameters in discrete steps.

Neither of these example collaboratories is an interactive application in the sense we use to describe VE, i.e. they do not accept continuous streams of input data and provide almost instantaneous feedback. The user interfaces are a command line or a collection of 2D widgets, manipulated without immediate feedback from the application. Humans tolerate a great deal of latency in interfaces of this type, so they are suitable when latency cannot be controlled.

3. Our Application

The nanoManipulator is a software and hardware tool that provides interactive 3D visualization of data from an Atomic Force Microscope and gives a user force-feedback control of the microscope's tip. The program allows scientists to see, feel, and modify samples ranging from DNA and carbon nanotubes to viruses and cells. All data coming from the microscope is recorded. This allows scientists to replay and reanalyze experiments at some time after they were originally conducted [1].

3.1. Collaboration System Requirements

The first priority in the Collaborative nanoManipulator system was to support the cognitive work of collaborative scientific research; secondly, the system needed to allow collaborators to be aware of one another's actions and to communicate [2]. The shared nM application is the heart of the system. As stated earlier, users expect the shared system to exhibit the same responsiveness as the single user system. Response time and user-to-user time should be under 300 ms. Our observations of the scientists' work patterns revealed that the system needed to support both working individually and working collaboratively (private and shared work), and needed to be able to switch back and forth easily between those two modes.

3.2. The Collaboration System

As it is the most important element of the system, we chose to devote most of our development resources to the shared nM application. We used off-the-shelf collaboration support products to enable visual and audio communication between the users and to allow sharing analysis tools and other productivity applications.

Figure 1 shows the collaboration system. One PC runs the shared nanoManipulator application with its attached Phantom™ force-feedback device; the second PC runs Microsoft NetMeeting™ for video conferencing and shared application support. We used an ordinary telephone, with wireless headset, to ensure high quality, full duplex audio communication. A drawing tablet was available, but seldom used. The scientists like having two video cameras, one a "talking head" view and one on a gooseneck that they could direct at their lab notebooks to share sketches or at their hands or apparatus. The local user selected which video signal to broadcast.

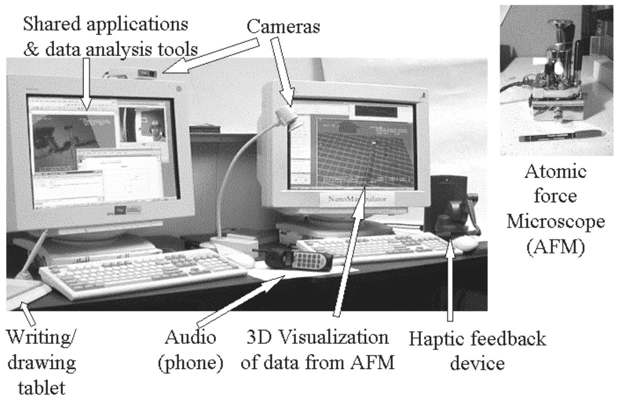


Figure 1. The Collaborative nanoManipulator System.

When using the Collaborative nanoManipulator System, two scientists geographically remote from one another can connect to the same Atomic Force Microscope. They receive the same data from the AFM, and can choose to link their visualization control parameters so that they see the data in the same way. This lets them discuss an experiment in progress or review a past experiment with a common frame of reference, lets them point to a region of the sample under the microscope and be sure that their collaborator sees the same thing they do, and lets them demonstrate data analysis techniques to one another.

4. Shared nM Architecture

Based on the need for very low latency to meet our application's interactivity requirement, our first design decision was to build a fully replicated system (see Section 2.2). We then applied the Model-View-Controller paradigm to structure our application replicas.

4.1. The Model-View-Controller Paradigm

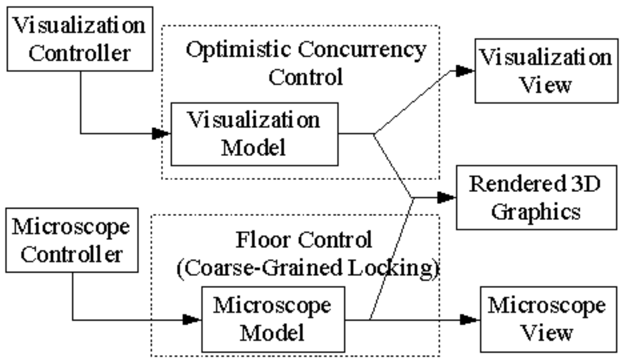
The Model-View-Controller (MVC) paradigm was first introduced as the standard architecture for applications in the Smalltalk programming language and has since been widely used to build many types of interactive applications, including collaborative tools ("groupware"). It specifies two facets of an architecture: the division of function among modules and the pattern of communication between modules. Application data and logic are grouped together in the Model, input-handling functions into the Controller, and output into the View. The Model notifies both View and Controller if any of its data changes, and they are responsible for determining whether or not this change is relevant to them. This generally inefficient communication pattern has been extended and optimized by several groups [19], until the MVC paradigm has come to describe the tripartite architectural decomposition of an application but not necessarily its communication pattern. It is in this more general sense that we use it. For clarity, Model, View, and Controller are capitalized in the following discussion.VEs deal with many different kinds of data. In CVEs, these different kinds of data typically have different requirements for concurrency control. In designing a CVE, we have found it useful to think of the application as not having a single instance of Model, View, and Controller, but as having several Models, whether in parallel or in a hierarchy, each with its own View and Controller. Each MVC triple can have its own concurrency control method. The View and Controller exposed to a user might be the union of all the instances, or a subset of them, with other Views and Controllers entirely internal to the application.

For example, in the Collaborative nanoManipulator System, there is a replicated Model of the microscope. This Model contains such information as, "What is the current location of the microscope tip? What is the current force exerted by the microscope on the sample? What is the topography of the surface recently scanned?" To avoid damage to the microscope, the Control of the microscope cannot permit any concurrent changes in the microscope control parameters. A locking mechanism is required because two users trying to direct the tip of the microscope at the same time while it interacts with a sample could cause damage to both microscope and sample and would guarantee that neither user's intention would be satisfied. For the AFM, and more generally for shared teleoperators, explicit locking/floor control is necessary.

The way the data in the Model of the microscope is visualized (View) is controlled by a complex set of parameters, which form a second, subsidiary Model that we'll call the visualization Model. The data in the visualization Model answer such questions as, "From what position (in a virtual space) is the user viewing the sample? What color map is applied to the sample? How is it illuminated? Are isocontour lines displayed?" Control for the visualization Model is very latency-sensitive (from a human-factors standpoint). The operations in the visualization Control are easily commutable and non-critical. The latency requirement and low probability of causing an inappropriate application response suggest that the visualization Model can be treated with optimistic concurrency control.

The data in the visualization-Model defines the difference between shared and private work: if two collaborators keep their visualization-Models synchronized, they see the microscope data in the same way; if not, they work independentlySeparating a set of requirements into distinct Models along the borders between feasible concurrency control schemes both helps us determine the architecture of the overall program and helps us build in flexibility. The graphical display of the microscope data in the Collaborative nanoManipulator is normally shared, but can be uncoupled so that each user has a different View. We accomplish this by uncoupling the visualization-Models; nothing in the microscope's Model-View-Controller system changes.

Figure 2. Two model-view-controller triples capture the different semantics of information about the microscope and about how the users wish to view that information.



We can also use this analysis to understand design decisions made in NPSNET, a system for immersive military combat training [16]. Although NPSNET is very different from the nanoManipulator, it still holds that different consistency requirements hold for different operations. NPSNET tracks the movement of players approximately, using low-bandwidth, optimistic techniques to keep the model at the player's local replica updated, with a recovery or convergence algorithm to handle errors caused by optimism [20]. When a player is shot at, the simulation requires a globally consistent answer; the decision of whether or not a shell hits is executed at one host and broadcast to all other interested participants. We can think of NPSNET as dividing its world data into two peer Models, each with an appropriate concurrency control method. Like the nanoManipulator, NPSNET uses optimism for speed but centralization for operations that cannot be undone.

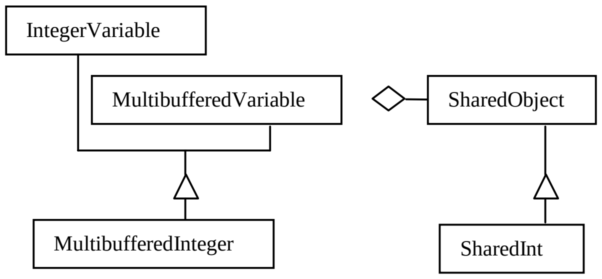
4.2. Implementing Hierarchical Models

For its visualization-Model, the nanoManipulator uses a fine-grained object-oriented design, where one heavyweight object exists for each primitive value (integer, float, string). A callback system propagates changes of the object's value to all interested entities.

We subclassed these objects, adding to every instance one buffer for each mode (private or shared). All concurrency control and networking concerns were handled by these buffers, so that we only needed to add to the interface of the primitive value objects the ability to select which mode was active for that object. This separation of concerns was very valuable in debugging and maintenance. When the user changes their collaboration mode, the value in an object's buffer for that mode becomes the object's value, triggering the object's associated callbacks.

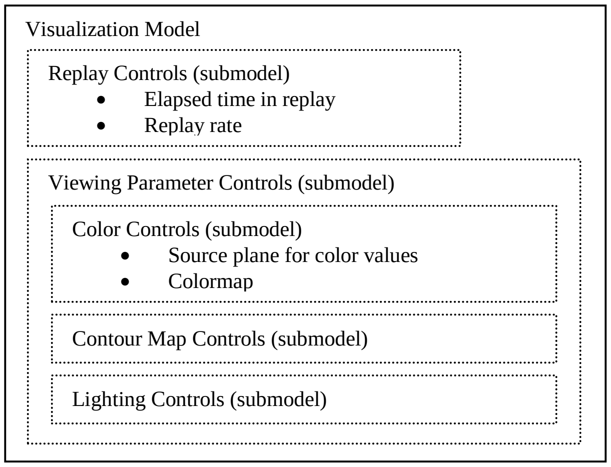
Thus, instances of the preexisting IntegerVariable class were replaced with MultibufferedInteger, which inherited both from IntegerVariable and a new utility class named MultibufferedVariable. MultibufferedVariable abstracted out the type-independent requirements of managing multiple buffers and switching between them. It contained two SharedObjects; for a MultibufferedInteger, these were instantiated as SharedInts. SharedObject and its descendants are classes that know how to keep one value synchronized over a network connection; which value is actually used at any point in time is controlled by the Multibuffered containers (Figure 3).

Figure 3. UML diagram of our implementation.



We then created a hierarchy of these objects from what had previously been a "sea" of global variables, each representing one primitive value (Figure 4). Using the hierarchy to group related interface parameters together increased the maintainability of the code. It also let us present meaningful chunks of the user interface coherently to the users.

Figure 4. Some of the hierarchy of objects in the visualization Model.


Visualization Model
Replay Controls (submodel)
• Elapsed time in replay
• Replay rate
Viewing Parameter Controls (submodel)
Color Controls (submodel)
• Source plane for color values
• Colormap
Contour Map Controls (submodel)
Lighting Controls (submodel)

We have experimented with using this hierarchy to allow fine-grained selection of which mode is active, so that a user can choose to have some parameters of their View shared with a collaborator while keeping others independent. In practice, we have found little demand among our users for this kind of mixed View. However, we think it could be quite relevant if, for example, a color-blind user wanted to use alternate visualization techniques.

Although the fine-grained design of the nanoManipulator allowed extensive use of a few base classes, and was an excellent design for the original single-user application, when extended to a distributed application it showed several shortcomings. There are a number of complex objects composed from these primitive value objects; for example, the orientation of the data is a quaternion but is implemented as four independent floating-point numbers. Changes to several of the values comprising a single complex object happened in a coordinated manner within a single process, but when transmitted across the network occurred as several distinct changes to the remote copy of the complex object. This caused both unnecessary network traffic and a good deal of difficulty with the design of the callbacks controlling these complex objects.

A system that would support composition of primitive value objects into complex objects would provide cleaner sharing. This coarser-grained system would not invalidate any of our results or proposed architecture. However, it could violate the separation of concerns (the “layering”) that placed mode selection in the primitive value object and synchronization in the per-model buffer, since the network synchronization would need to know about the embedding of the primitive value object into a larger construct.

4.3. Synchronizing Multiple Models

We used optimistic concurrency control to keep consistent replicated state in the visualization-Models (Table 1). Although optimistic protocols are intended to provide fast concurrency control, their speed is dependent on the speed of the serialization algorithm they use to agree on the order in which events happen at the nodes of the distributed system. As did NPSNET and AVIARY, we found that wallclock serialization [21] works extremely well for shared VE.

Table 1. Synchronization techniques appropriate for various scenarios.

Technique	Suitable Scenarios
Coarse-Grained Locks	Objects to be acted upon are interdependent, or are considered to be interdependent by users constructing plans
Fine-Grained Locks	Objects to be acted upon can be separated into independent subsets
Explicit Locks	Not losing work is critical, even at the cost of latency and a changed workflow
Implicit Locks	Not interrupting natural workflow is critical, even at the cost of latency or lost work
Optimism	As fine-grained, implicit locks; ideally, conflicting user actions can be automatically reconciled; little risk of damage, easy to undo

We used the Virtual Reality Peripheral Network protocol [22] as the Session layer for the Collaborative nanoManipulator, giving us the ability to make asynchronous remote procedure calls (RPC). With asynchronous RPC, when a process requests an operation from another process, the requesting process does not wait for values to be sent back by the remote server, but instead resumes execution immediately. Receipt of the response from the remote server will trigger a callback in the requesting process. This non-blocking communication helps to decouple the application from network latency, allowing an interactive application to reduce response time for operations that are being carried out across the network. In this way, asynchronous RPC is more useful for interactive applications than is traditional synchronous RPC. Asynchronous RPC typically leads to a more complex programming style centered on callbacks; however, this same style is also required by modern graphical user interfaces, and was adopted for most of the Collaborative nanoManipulator.

5. Evaluation

5.1. Concurrency Performance

The Collaborative nanoManipulator was originally deployed on dual-processor 500 MHz Windows NT workstations machines on a 100 MB switched Ethernet LAN. While we expected concurrency control to be a problem when we extended the system across the Internet, we were surprised to find that even on a single campus the latency incurred from even simple implementations of non-optimistic concurrency control were prohibitive. Details follow.

Deployed on the hardware outlined above, a single user of the system saw a 214 ms response time. Our original collaborative implementation used a server to serialize every operation on the visualization-View parameters, guaranteeing consistency. An arbitrary instance of the application was selected as the serialization server. The user at the server experienced essentially single-user response time, since there was no need to wait for a network round-trip to serialize their actions. However, the remote collaborator saw 319 ms of additional latency for every manipulation. Since this is in addition to the system's native 214 ms response time, remote collaborators strongly objected to the system's responsiveness, refusing to use the collaborative system.

The server-based serialization was significantly slowed by the fact that it was running within the nanoManipulator process, limited by that process' 214 ms response time. On computers with fast graphics hardware (workstations with nVidia Quadro Pro 2 graphics), the single-user response time drops below 30 ms. The 214 ms of server response time could also be removed from serialization costs by a multithreaded implementation or a dedicated server. However, any of these approaches — multithreading, a dedicated server, or newer hardware — would still add an unacceptable 105 ms of latency to the application's base response time.An improvement might be to use implicit fine-grained migrating serialization, so that the latency penalty only needs to be paid once for each manipulation. However, this is a complex approach that does not completely eliminate serialization latency and can degrade poorly. It is also not compatible with our goal of composing very-fine-grained objects. Our solution was to use wall-clock serialization. Instead of implementing a complex distributed algorithm, we installed off-the-shelf software on all of our computers that keeps their clocks synchronized with the US government's official atomic clock. We observed that the clocks typically stayed synchronized to within 10 ms. Since there was no overhead from clock synchronization, this gave us performance similar to single-user mode and satisfied our users (Table 2). In practice, even large errors in clock synchronization do not lead to a visibly inconsistent ordering of events.

Table 2. Performance of concurrency control.

Mode	Mean Response Time (ms)	Mean Response Time due to Concurrency Control (ms)
Single-user (baseline)	214	--
Server-based serialization	533	319
Wall-clock optimism	262	48

5.2. Collaboration System Evaluation

A report on a repeated-measures, controlled experiment evaluating the collaborative nanoManipulator is in [23]. 20 pairs of upper-level undergraduate science majors participated in two lab sessions, one session face-to-face using the standard nanoManipulator and the other session using the Collaborative nanoManipulator. As expected, participants reported disadvantages to collaborating remotely. When working remotely, interaction was less personal, individuals received fewer cues from their partner, and some tasks, such as sharing math formulas, were more difficult. However, participants also reported that some of these disadvantages are not significant in scientific work contexts, and that coping strategies, or work-arounds, can reduce the impact of other disadvantages.

Participants reported that remote collaboration provided several advantages compared with face-to-face collaboration, including the ability to more easily explore the system and their ideas independently and increased productivity with the ability to work simultaneously on the data visualization. Analysis of graded lab reports created under both conditions showed no statistically significant difference due to condition.

The study participants were asked what they like and didn't like about the system in post-experiment interviews. Of particular interest to the work reported in this paper, many participants reported that they liked the ability to simultaneously adjust visualization-Model parameters when in shared mode and found that using the explicit floor control in shared applications running in NetMeeting hindered their work.

6. Conclusions

Building the Collaborative nanoManipulator, we found that the model-view-controller paradigm was a useful way to analyze CVEs. Extending model-view-controller to include multiple parallel or hierarchical Models highlights the differing concurrency control requirements of different subsets of an application's state data and helps us design the application's architecture.

To support the interactivity requirements of our CVE, we used asynchronous RPC and optimistic concurrency control. Optimism is a valuable technique particularly well suited for minimizing latency of continuous, easily reversed inputs to a VE, such as a users' position and orientation. It is less amenable to supporting transactions and complex assemblies of primitive operations, but can be used to do so with a moderate software engineering effort.

All of these techniques – from high-level analysis and user interface design down to details of implementation – should prove helpful for other groups implementing a wide variety of CVE systems.

7. Acknowledgements

More than a dozen computer and information scientists worked on the nanoManipulator project during the development and initial deployment of the system; in particular, Kelly Maglaughlin, Eileen Kupstas Soo, and Ron Bergquist contributed to the design of the user requirements. Russell Taylor and the anonymous reviewers had suggestions that strengthened the paper. We must also thank the many physicists, chemists, and biologists who have worked with the nanoManipulator project, and the NIH National Center for Research Resources and the National Institute for Biomedical Imaging and Bioengineering (Grant P41 RR 02170).

8. Bibliography

[1] R. M. Taylor and R. Superfine, "Advanced Interfaces to Scanning Probe Microscopes," in *Handbook of Nanostructured Materials and Nanotechnology*, H. S. Nalwa, Ed. New York: Academic Press, 1999, pp. 271 - 308.

[2] D. H. Sonnenwald, R. E. Bergquist, K. L. Maglaughlin, E. Kupstas Soo, and M. C. Whitton, "Designing to Support Collaborative Scientific Research Accross Distances: The nanoManipulator Environment," in *Collaborative Virtual Environments*, *CSCW*, E. F. Churchill, D. N. Snowdon, and A. J. Munro, Eds. London: Springer-Verlag, 2001, pp. 202 - 224.

[3] G. E. Krasner and S. T. Pope, "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80," *JOOP*, pp. 26 - 49, 1988.

[4] S. Bhola, G. Banavar, and M. Ahamad, "Responsiveness and Consistency Tradeoffs in Interactive Groupware," presented at CSCW, 1998.

[5] S. Bryson and S. Johan, "Time Management, Simultaneity and Time-Critical Computation in Interactive Unsteady Visualization Environments," presented at Proceedings of IEEE Visualization, 1996.

[6] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz, "NPSNET: A Network Software Architecture for Large-Scale Virtual Environments," *Presence*, vol. 3, pp. 265 - 287, 1994.

[7] C. Ware and R. Ralakrishnan, "Reaching for Objects in VR Displays: Lag and Frame Rate," *Transactions on Computer-Human Interaction*, vol. 1, pp. 331 - 356, 1994.

[8] C. D. Wickens and P. Baker, "Cognitive Issues in Virtual Reality," in *Virtual Environments and Advanced Interface Design*, W. Barfield and I. Furness, Thomas A, Eds. New York: Oxford University Press, 1995, pp. 514 - 541.

[9] I. Vaghi, C. Greenhalgh, and S. Benford, "Coping with Inconsistency due to Network Delays in Collaborative Virtual Environments," presented at VRST 99, London, UK, 1999.

[10] M. Herlihy, "Concurrency versus Availability: Atomicity Mechanisms for Replicated Data," *Transactions on Computer Systems*, vol. 5, pp. 249 - 274, 1987.

[11] R. Malpani and L. A. Rowe, "Floor Control for Large-Scale MBone Seminars," presented at ACM Multimedia, Seattle, WA, 1997.

[12] M. Herlihy, "Apologizing Versus Asking Permission: Optimistic Concurrency Control for Abstract Data Types," *Transactions on Database Systems*, vol. 15, pp. 96 - 124, 1990.

[13] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhauser, "An Integrating, Transformation-Oriented Approach to Concurrency Control and Undo in Group Editors," presented at CSCW, Cambridge, MA, 1996.

[14] M. Meehan, "Survey of Multi-user Distributed Virtual Environments," In course notes: "Developing Shared Virtual Environments". *SIGGRAPH 99*. Los Angeles, CA. August 8-13, 1999.

[15] D. Snowdon and A. West, "Aviary: Design Issues for Future Large-Scale Virtual Environments," *Presence*, vol. 3, pp. 288-308, 1994.

[16] M. R. Macedonia, D. P. Brutzman, M. J. Zyda, D. R. Pratt, P. T. Barham, J. Falby, and J. Locke, "NPSNET: A multi-player 3D virtual environment over the internet," presented at I3D, 1995.

[17] C. S. Potter, B. Carragher, L. Carroll, C. Conway, B. Grosser, J. Hanlon, N. Kisseberth, S. Robinson, U. Thakkar, and D. Weber, "Bugscope: A Practical Approach to Providing Remote Microscopy for Science Education Outreach," Beckman Institute, UIUC, Urbana, IL 1 November 2000 2000.

[18] G. M. Olson, D. E. Atkins, R. Clauer, T. A. Finholt, F. Jahanian, T. L. Killeen, A. Prakash, and T. Weymouth, "The Upper Atmospheric Research Collaboratory," in *interactions*, 1998, pp. 48 - 54.

[19] C. Schuckmann, J. Schummer, and P. Seitz, "Modeling Collaboration using Shared Objects," presented at Groupware, Phoenix, AZ, 1999.

[20] S. Singhal and M. J. Zyda, *Networked Virtual Environments: design and implementation*: ACM Press, 1999.

[21] T. C. Hudson, "Concurrency Control for Collaborative 3D Graphics Applications," University of North Carolina, Chapel Hill UNC CS TR01-021, 2001.

[22] R. M. Taylor, T. C. Hudson, A. Seeger, H. Weber, J. Juliano, and A. T. Helser, "VRPN: A Device-Independent, Network-Transparent VR Peripheral System," Proceedings of ACM Symposium on Virtual Reality Software & Technology 2001, Banff Centre, Canada.

[23] D. H. Sonnenwald, M. C. Whitton, and K. L. Maglaughlin, "Evaluating a scientific collaboratory: Results of a controlled experiment," *in review*, 2002.