Handling Heterogeneity in Networked Virtual Environments

Helmuth Trefftz
htrefftz@sigma.eafit.edu.co
Universidad EAFIT
Computer Science Department
P.O. Box 3300
Medellin, Colombia

Ivan Marsic
marsic@caip.rutgers.edu
Rutgers University
Department of Electrical and Computer
Engineering and the CAIP Center
96 Frelinghuysen Road
Piscataway, NJ 08854

Michael Zyda
zyda@movesinstitute.org
Naval Postgraduate School
The MOVES Institute
833 Dyer Road, Spanagel Hall 252
Code MOVES/mjz
Monterey, CA 93943-5118

Abstract

The availability of inexpensive and powerful graphics cards as well as fast Internet connections make networked virtual environments viable for millions of users and many new applications. It is therefore necessary to cope with the growing heterogeneity that arises from differences in computing power, network speed, and users' preferences. This paper describes an architecture that accommodates the heterogeneity while allowing a manager to define systemwide policies. One of the main objectives of our scheme is to allow slower nodes to participate in the session by preventing fast nodes from flooding slow nodes with too many messages. Policies and users' preferences can be expressed as simple linear equations forming a model that describes the system as a whole as well as its individual components. When solutions to this model are mapped back to the problem domain, viable solutions that accommodate heterogeneity and system policies are obtained. For example, slower nodes may receive less frequent updates than faster ones for one or several information streams. The results of our experiments with a proof-of-concept system are described.

1 Introduction

Powerful computer graphics cards are becoming less expensive, and most new computers come equipped with cards that are capable of displaying complex 3D scenes at interactive rates. Similarly, high-speed connections to the Internet are becoming increasingly common. These developments allow for the potential widespread use of distributed virtual environments or networked virtual environments (NVEs) (Singhal & Zyda, 1999).

In a controlled environment, the NVE designer can determine, a priori, the type of computers and network connections that will be involved in the system. The NVE can then be engineered to perform well within the given constraints. But, as NVEs find their ways to real-world applications, the environments cannot be so easily controlled. New computers are often mixed with older machines, and significant heterogeneity naturally arises, as illustrated in figure 1. Heterogeneity arises in two different forms:

- nodes with a wide variety of computing resources (processor speeds, available memories, and graphic cards), and
- varying network connection type (from modern lines to local area networks).

Presence, *Vol. 12, No. 1, February 2003,* 38–52
© 2003 by the Massachusetts Institute of Technology

The inequity of the machines in an NVE session makes the machines with lesser resources vulnerable to large amounts of information generated by high-

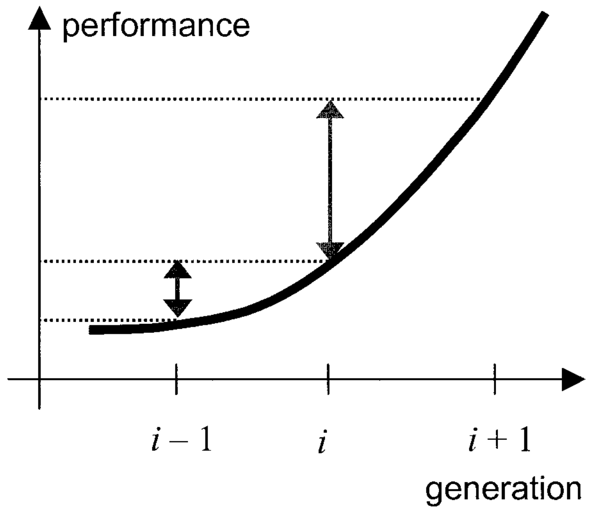

Figure 1. *According to Moore's Law, computer performance grows exponentially. So does heterogeneity between the generations, as the curve shows.*

end machines. For example, in a scientific visualization session, lower-end machines can become saturated by messages from fast machines. This results in degraded user experience, or the machine simply stops responding to the local user's input. It is necessary to protect the less capable machines from being overwhelmed.

In addition, designers of NVEs need to be able to determine certain systemwide constraints to guarantee productive use of the system. Examples of these policies include the maximum number of concurrent users, the minimum framerate that each node must be able to maintain, the maximum rate of messages that can be handled by the system, and so forth.

On one hand, the design must support the heterogeneity that arises from the diverse participating nodes and from individual user's preferences. On the other hand, the design should provide the system administrator with the ability to dictate and enforce a certain homogeneity of function.

In this paper, we formulate the problem of satisfying the user and global constraints with heterogeneous resources as an optimization problem. The available resources determine the performance space of each computing node. The individual user preferences and global system constraints are expressed as linear equations and inequalities. This set of equations and inequalities forms a mathematical model that captures system resources, user preferences, and global constraints. Valid solutions of the model are isomorphic with states of the NVE in which individual user's preferences are satisfied as closely as possible, given the restrictions imposed by the resources and the constraints defined by the NVE administrator. The algorithm produces the solution that maximizes the objective function of the model. We also propose a client-server architecture that we call switchboard architecture (SA) as a way to implement the model.

The rest of this paper is organized as follows. Section 2 describes the generic form of the mathematical model as well as the corresponding switchboard architecture. Section 3 and 4 describe the experimental setup to evaluate the model and the results we have obtained with a shared visualization application. Section 5 describes related work, and section 6 presents the conclusions and future work.

2 The Switchboard Architecture

We assume a client-server architecture, with one or more servers supporting collaboration of multiple clients. Some shared data in a virtual environment may originate or be cached locally whereas other may originate from remote and change with time. The data that originates locally may need to be distributed to other participants. The clients have local computing resources and share some global resources, such as server(s) and network bandwidth, which support collaboration. Our objective is to satisfy the user and global constraints with heterogeneous resources, and we formulate this as an optimization problem. We first consider individual clients as data consumers that need to visualize the data with the best quality under the given constraints. Then we consider the clients additionally as data producers that need to update the other participants by complying with the constraints on global resources.

2.1 Mathematical Model

Different qualities or dimensions of shared data in a virtual environment represent variables, examples of which include the update rate for time-dependent data (video framerate), the visualization fidelity (such as wireframe or shaded), and the sound fidelity (such as mono, stereo, or 3D). To achieve a certain value of a quality variable requires certain amounts of computing power dedicated to processing the related data. The qualities that are specified externally to the system as requirements are independent variables. In addition, we have system performance variables, such as visualization framerate or number of dropped messages from remote nodes. Performance variables are determined by system resources and take values depending on the independent variables; they are thus dependent variables.

We assume that the variables take on discrete values within a certain range, determined by the computing resources of the nodes taking part in a session. For each node, the Cartesian product of the possible values of the different variables forms an *n*-dimensional performance space, with *n* being the number of variables.

Our approach works by controlling the independent variables and letting the operating system (OS) allocate all the remaining resources to the dependent variables. The amount of resources allocated to the dependent variables depends on the total amount of resources at the node, such as CPU speed, specialized processors, and network bandwidth. The available physical resources are also affected by the choice of OS, programming language, and so on. The (nonlinear) relationship between the variables at a node *k* is called the performance mapping:

$$E_j = f_{kj}(I_1, I_2, \ldots, I_{nj}), \quad j = 1, \ldots, n_E, \tag{1}$$

where n_I is the total number of independent variables and n_E is the total number of dependent variables, $n_I + n_E = n$. The mappings f_{kj} are node specific, and the node resource characteristics are embedded in the mappings. They convey the fact that dependent and independent variables compete for the same resources. The performance mapping is determined experimentally by a set of benchmarks that are run at each participating node before starting a collaborative session.

Global policies can be expressed as inequalities over certain independent and/or dependent variables. An example of a global policy might be "Each participating system must be able to display at least two video images per second."

An individual's preferences can be expressed as a linear combination of the variables, with the normalized coefficients determining their relative importance. The normalization must be done carefully, taking into account the ranges of the corresponding variables.

Global policies partition the performance space into valid subspaces. The intersection of these valid subspaces is, in turn, either a finite or an infinite *n*-dimensional space. To enforce the global policies, variables at each node are limited to taking only values that lie inside the intersection of valid subspaces. The intersection of valid subspaces can be empty, for instance, when a node is not able to perform at the minimum levels defined by the administrator or the user's preferences cannot be met with current resources. In this case, the user may try to restate his or her preferences or the node cannot be admitted to the session.

The linear combination of variables is the objective function, which measures the user satisfaction, since its maximum best meets the user preferences under the given constraints. The optimization problem at a node *k* is as follows.

$$Max \left\lfloor \sum_{i=1}^{n_I} (W_{ki}^I \times \hat{I}_i) + \sum_{j=1}^{n_E} (W_{kj}^E \times \hat{E}_j) \right\rfloor, \tag{2}$$

subject to:

$$I_i \geq K_i, \quad i = 1, \ldots, n_I \tag{3}$$

$$E_j \geq L_j, \quad j = 1, \ldots, n_E \tag{4}$$

$$I_i \leq \mu_i, \quad i = 1, \ldots, n_j, \tag{5}$$

where $\hat{I}_i$ is the normalized i_{th} independent variable,
$\hat{E}_j$ is the normalized j_{th} dependent variable,
W_{ki}^I, W_{kj}^E are the weights representing the relative importance of the associated variables according to the user *k* preferences,

K_i and L_j are the minimum values that the i_{th} independent variable and j_{th} dependent variable can assume across all the clients, respectively, and

μ_i are constraints that may be imposed on the upper values of some independent variables I_i as will be explained in a moment.

The model of equation (2) through (5) is optimized independently at each client site. The objective function (2) incorporates the k_{th} user's preferences and the local resources through the relationship (1). Global constraints on variable values (equation (3) through (5)) delimit the valid search space. The objective function is evaluated at each point of the valid search space looking for a maximum (exhaustive search). The optimum values of the dependent variables are assumed to be the results of the performance mapping at the point that maximizes the objective function. The values of the variables at the point at which the objective function is maximized correspond to the fidelity level that the client application must set for independent variables to best adapt to the user's preferences while complying with the global policies.

The model is similar in form to a linear programming problem, but the model cannot be handled with linear programming techniques because the variables do not take continuous values and because the dependent variables are not linear with respect to the independent variables. However, given the small size of the search space, each client can independently apply an exhaustive search and evaluate the objective function at each valid point to quickly find the maximum. Problems involving more variables or fidelity levels may use a different search algorithm to avoid an exhaustive search. Each optimization is performed locally, and this is what makes the model scalable to large NVEs.

The model so far considers each client independently of others as a data consumer, which selects the most-suitable fidelity levels from those offered to meet the constraints and maximize the user satisfaction. We now consider the client additionally as a data producer that needs to update the others by using the shared resources.

At present, the only dimension of the update messages that is considered is the message rate. Other dimensions, such as message data fidelity or different communication protocols, are not currently considered. Let J_i denote the independent variables that represent the update rates for different shared data types, and n_J is the total number of such variables. Global constraints limiting the maximum rate of messages the server can process have the following form:

$$\sum_{k=1}^{N} J_i(k) \leq M_i, \quad i = 1,\ldots,n_J, \tag{6}$$

where N is the number of participants in the NVE,

$J_i(k)$ is the rate of type i messages generated by client k, and

M_i is the maximum message rate of type i that the server can process.

The upper limits μ_i in equation (5) can be derived from M_i and one way of doing this is presented as follows.

Our approach works in two phases:

- *Offline*—Prior to actively joining the session, each node runs a series of tests to determine the performance mappings (equation 1). During the tests, the node is set to work under the Cartesian product of the independent variables, and the values of the dependent variables are measured and recorded. Thus, the search space is predetermined for each node. It is assumed that every node's behavior will remain relatively constant.
- *Online*—During the collaborative session, the system needs to adapt to changes that may occur. For instance, the user might change her preferences, thereby changing the relative importance of the variables. A new search for the maximum value inside the local valid search space has to be conducted. Or, a new user might join the session, adding to the number of messages the server has to process. Because the behavior of the client can be erratic, the frequency of dynamic adaptation must be constrained. Otherwise, the optimization process might take too much computing power and affect the collaborative experience. In our current implementation, 10 sec. must elapse between the consecutive runs of the adaptation algorithm.

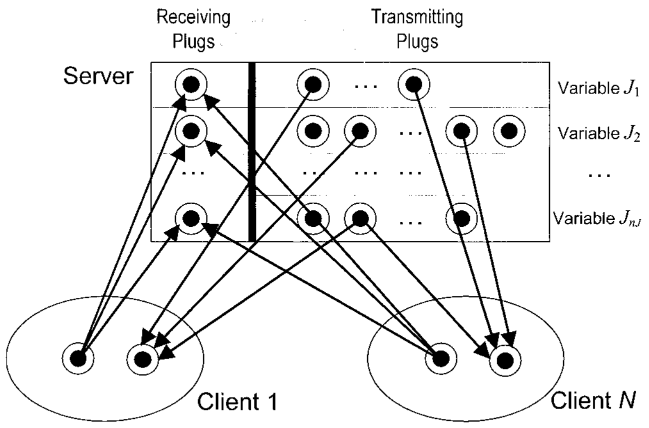

Figure 2. *The switchboard message distribution architecture. Each plug corresponds to a specific instance of an independent variable and its representation fidelity.*

2.2 The Switchboard Architecture

The model does not impose a particular implementation. Here we present an architecture, called switchboard architecture (SA), as a potential implementation of the model. (The architecture is shown in figure 2). Each row of the switchboard handles an independent variable in the system. Updates for a specific variable from all the participating clients are received in a receiving plug, and the server distributes those updates into the switchboard area. The elements of the switchboard area represent the transmitting plugs. They correspond to different fidelities of the shared information according to client capabilities and user preferences. (Note that the switchboard area is not necessarily rectangular because different variables may have different numbers of fidelity levels.) Each transmitting plug has an associated timer controlling its transmission period. When this timer expires, all the updates stored in the plug's cache are transmitted to the subscribed clients (if any). To reduce traffic, each transmitting plug may be implemented as a multicast group. Also, instead of each plug in the same row having different rates (simulcast), we could implement layered multicast (Vickers, Albuquerque, & Suda, 2000). To improve scalability when large number of multicast groups is required, multicast group clustering can be employed (Riabov, Liu, Wolf, Yu, & Zhang, 2002).

The reason for choosing a client-server architecture is that the optimization model includes monitoring global resources. It is possible to collect this information in a peer-to-peer system, but this would imply either adding an agent at each node or adding a specialized node to monitor the activity in the system. We chose a client-server architecture that gives us the possibility of gathering all the necessary statistics at a centralized point. Our focus is on small-scale collaboration systems (tens of participants), for which client heterogeneity is a greater problem than server performance. The solution can be generalized to larger collaborative systems by using different servers for different variables or even multiple servers, each for different fidelity of a variable. This requires further investigation to assess the scalability of the solution.

The switchboard architecture is middleware that manages the use of heterogeneous resources under possibly conflicting user interests and global policies. The characteristic of this architecture is that the server acts as a buffer, providing slower nodes with a subsample of messages generated by faster nodes in a controlled manner that is determined by the client node's resources and user preferences. The apparent jumpiness of the objects that end up being updated less frequently can be compensated, to an extent, with dead reckoning. We provide more details and describe our particular implementation in the following section.

3 Experimental Setup

As a proof of concept for the architecture and the model, a shared visualization system was built using Java3D (Sowizral, Rushforth, & Deering, 1997). Figure 3 shows a screenshot of the client interface. Three users meet virtually to discuss over a shared visualization data set. Users utilize telepointers (3D arrows) as means to point inside the virtual world, and small video windows are provided to allow users to see the faces of all participants. The sliders allow the user to set the relevance vector that expresses the user's preferences, used as weights W in equation (2). Only one user can manipulate the visualized object at a given time. The applica-

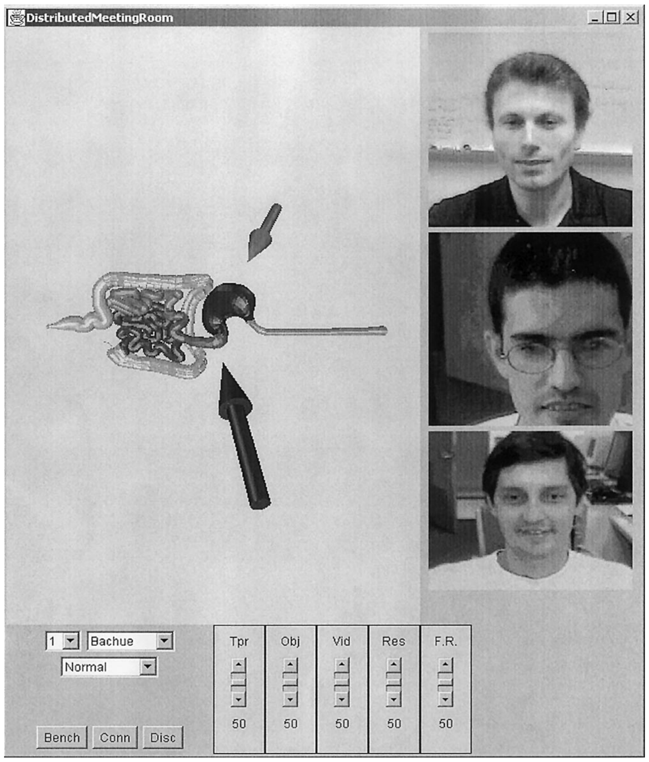

Figure 3. *Screenshot of the client. Telepointers are represented as 3D arrows. The sliders allow the user to set the relevance vector of the variables.*

tion directly handles all the media related to the variables to be able to optimize the objective function. We could use third-party applications, such as those for video conferencing, if they provide APIs to control the level of fidelity and the statistics on the number of lost messages.

The administrator can determine the global constraints for each modality involved in the session. Global constraints are embodied in the following parameters.

- Minimum fidelity that each client node must be able to maintain, (K_i and L_j in equation (3) and (4)), such as a minimum framerate that each client must be able to sustain.
- Maximum acceptable percentage of dropped messages. For instance, it is not acceptable for any client to drop more than 50% of video messages. (Note that the messages may be lost in transmission and inside the application when the machine cannot keep up with the incoming flow of messages.)
- Maximum message rates on the server for independent variables, (M_i in equation (6)), such as the maximum number of video messages the server can receive and retransmit per second.

The first two constraints are enforced at each client. Whenever the client node is not able to maintain one of them, either because the network has become congested or because there is a new process running on the node, the optimizer module is invoked to search for a new solution that will allow the node to comply with the global constraints.

The last constraint is enforced by a combination of actions taking place at the server and at the client. The aggregate number of requested messages by each client cannot exceed the server capacity, that is, the total number of messages that the server can process.

We define the global upper bound, μ_i, as the maximum number of messages that a client, *k*, can request so that the aggregate requests do not exceed the server capacity, M_i. (See equation (5) and (6).) This constraint may be violated as a result of changes in the system, such as when a new user joins the session. In this case, the server recomputes and broadcasts a new global upper bound μ_i on the resolution for that specific modality that *each* client node can request. The new global upper bound is computed so that the aggregate resolution requested by all the clients does not exceed the server capacity. Client nodes that are subscribed to resolutions higher than the new global upper bound then need to lower their requests. Usually, nodes subscribed to lower resolutions do not need to change their subscriptions. See figure 4 for an illustration.

3.1 Variables

The variables (modalities) involved in the NVE system are as follows.

- *T:* telepointers update rate
- *O:* visualization data set update rate
- *V:* video update rate
- *G:* graphical representation fidelity

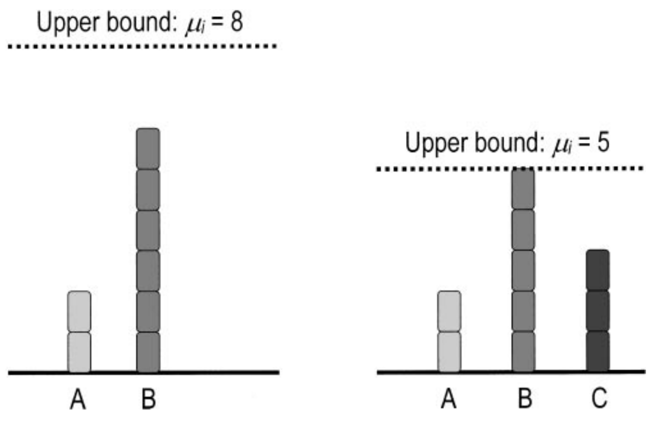

Figure 4. *The graph on the left represents the system constraint set for users A and B and a global upper bound $\mu_I = 8$. The figure on the right describes the changes that take place when user C joins the session requesting three messages/unit of time. The server capacity is $M_i = 10$.*

- *F:* scene visualization framerate
- *D_i:* percentage of dropped messages for an independent variable, J_i

Variables *T, O,* and *V* can take the following values: 0 (no updates), 1, 2, 10, or 20 updates per second. Variable *G* can take value 0 (wireframe representation) or 1 (Phong shaded). Another use of variable *G* could be different levels of detail of the visualization data set. Higher levels of detail correspond to larger values of *G*. Variables *T, O, V,* and *G* are independent variables, and variables *F* and D_i are the dependent variables. The values *F* can take are the result of the performance mapping, and they vary considerably across machines.

The sliders shown in figure 3 provide users with a way to assign relative weight to the variables. The values of the sliders at client *k* are normalized to obtain the coefficients for the objective function as follows.

$$W_{ki} = \frac{S_{ki}}{\sum_{j=1}^{n} S_{kj}}, \quad i = 1, \ldots, n, \tag{7}$$

where S_{ki} are the values of the sliders at client *k,* between 0 and 100, and W_{ki} are the values of the coefficients used in the objective function of client *k* (equation (2)).

Note that the selection of the variables for the model is specific to the application. For instance, in a battlefield simulation, one variable might be assigned to slow-moving vehicles, such as tanks and another to fast-moving vehicles, such as airplanes. Users of one type of vehicle will probably be more interested in (and will assign higher priority to) messages originating from vehicles of similar type. Another example is one in which video streams from different users are assigned different priorities, in which case we will have different variables for each video stream.

3.2 Scenario for Applying the Model

Here we describe a typical scenario for employing our model in a collaborative session. Before the session starts, we need to empirically determine the performance mappings for the dependent variables (equation (1)). We run a benchmark program on each node to create the data for its particular performance mapping. The node subscribes sequentially to every point in the Cartesian product of the dependent variables for a certain period (long enough to allow the behavior of the dependent variables to stabilize), which should be determined empirically. In our case, the main dependent variable is framerate, the required time is 10 sec., and there is a 10 sec. pause between the successive measurements to let the messages from the previous measurement clear. The dimension of the search space in our example is 250 ($|T| \times |O| \times |V| \times |G| = 5 \times 5 \times 5 \times 2$, where $|T|$ is the range of different fidelities that the telepointer data type can assume, and so on). During the benchmark, the server generates fake updates for telepointers, object movement, and video. Each message is marked with a sequence number, to detect lost messages. At the client, an entry with the current time in milliseconds is added to a vector every time a frame is displayed. This allows the computation of the average framerate for the particular search space vertex. Additionally, for each message that is processed, an entry is added to a vector with the message sequence number to count the number of lost messages at each search space vertex. When the benchmark finishes, the vectors are saved to disk and summarized in a performance map-

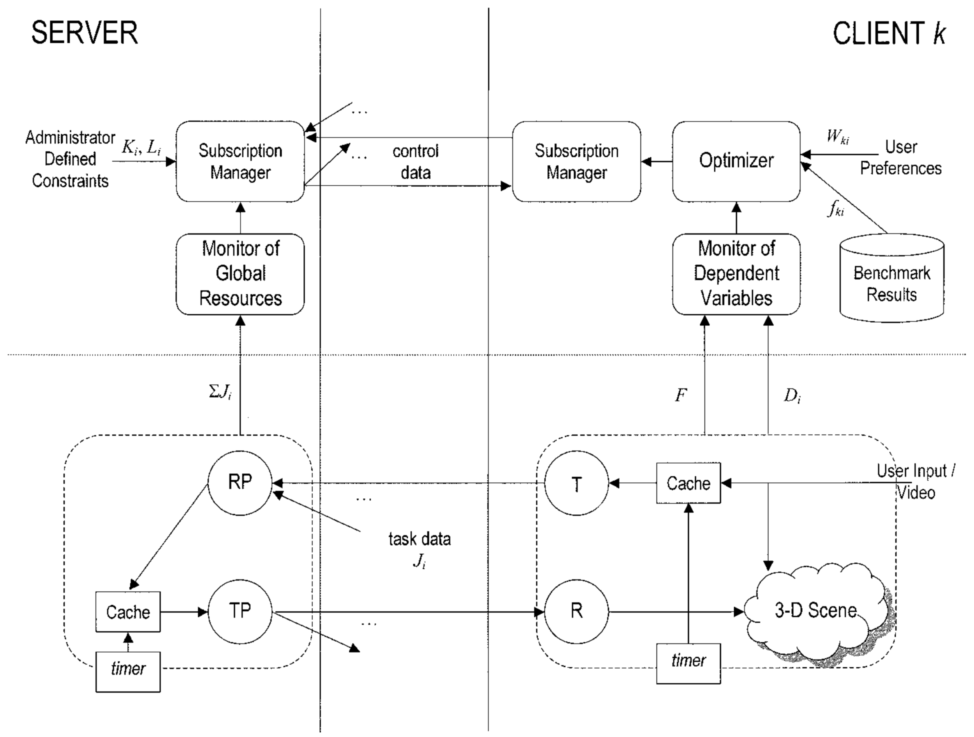

Figure 5. *Components of the switchboard architecture. The upper portion of the figure describes the flow of control data that control the system operation; the lower portion describes the flow of task data exchanged in the collaborative session. The variables are defined in equation (1) through (6). TP = transmitting plug, RP = receiving plug, R = receiver, and T = transmitter as described in figure 2.*

ping for the particular node. Next, the collaborative session starts.

Figure 5 shows the components of the proof-of-concept system. The lower portion describes the components involved in the exchange of actual data shared in the collaborative session. In our case this includes the telepointer, object, and video messages (audio was not used in this experiment). We use an unreliable protocol (UDP) to exchange data between the client and the server. As mentioned earlier, each transmitting plug is mapped to a multicast group.

The upper portion of the figure describes the exchange of metadata, that is, information on how the system runs. The two types of metadata messages are messages in which the client specifies to which multicast groups it wishes to subscribe, and messages in which the server broadcasts updates on the systemwide constraints. Due to the importance and low frequency of metadata messages, we use a reliable protocol (TCP) for their transmission.

When the session starts, the optimizer module in the client first finds the solution that maximizes the objective function that corresponds to all the sliders at 50 (the sliders can take values between 1 and 100). The client then subscribes to the server to the appropriate plugs and sets the fidelity of the visualization data set according to the current solution.

The optimizer is invoked in the following cases.

- when the client establishes a connection to the server for the first time;
- when the user updates the relevance vector by adjusting the sliders in her interface;
- when the administrator changes the values of global or local constraints;

- when any condition in the system implies changing the global constraints (for example, a new user joins the session and requests a message rate that causes the server capacity for that variable to be exceeded); and
- when the local monitor of dependent variables is triggered to update the performance conditions. (Currently, the system monitors the framerate and the drop rate for each modality. This way, optimization decisions are not made based on the benchmarks only, but also on dynamic performance data collected by the local monitor.)

In each of these cases, the client adjusts the model and searches for a new solution. The search involves evaluating the objective function at the valid vertices (250 or less because some of the search vertices are marked as invalid). As a result, it might be necessary to unsubscribe from a particular plug (associated with a fidelity level of a variable) and subscribe to a different one.

Subscription and unsubscriptions from the multicast groups are handled by the subscription manager modules, both at the server and the client. To comply with the global upper bounds, it is necessary to limit the maximum number of messages that a client generates per unit of time. If a client is subscribed to receive a particular frequency of updates for a specific variable, the same frequency is the upper limit to the number of updates that the client is allowed to generate. To enforce this, we cache the messages at the clients and thus limit the source rate. Note the caches and the corresponding timers, both at the client and at the server. This is necessary to enforce equations such as equation (6) and allows the system administrator to specify upper limits on the number of messages the server will need to process for each variable.

The exchange of actual collaborative data starts with the user input, which is reflected immediately in the local scene. The update message is stored in a local cache and sent to the server by the transmitting module (T) when the associated timer is triggered. It is received by the server in the receiving plug (RP), which stores a copy of it in a cache of all the transmitting plugs (TPs) for that modality. Previous messages of the same type originating in the same client are overwritten in the caches. When a timer associated with a plug expires, the message is multicast to the clients subscribed at that TP. Remote updates arrive at the client in the receiving module (R) and update the scene immediately.

The caching procedure resembles the leaky bucket algorithm used for traffic shaping in data networks (Tanenbaum, 2003).

3.3 Adaptive Behavior

Unless the application runs in a completely controlled environment, the conditions of the network and computer resources during the benchmarks and during the collaborative session will not be the same. More (or less) bandwidth will be available depending on the congestion of the network. Additionally, some unrelated processes might be running in the client computer during the collaborative session.

Our system has two ways of monitoring network and system activity:

- The visualization framerate is continuously monitored at the client machine. If another process is running on the client machine and affecting the performance, this will be reflected in the framerate.
- Each transmitting plug maintains a sequence number that is incremented whenever a message is sent to the network. The sequence number is included in the messages, thus allowing the clients to detect lost packets.

The monitor of dependent variables module at the client keeps track of the framerate and the percentage of dropped messages for each modality. Whenever the optimizer module is invoked, the monitor provides the current values for framerates and drop rates. The optimizer module uses these data to extrapolate the values of the dependent variables (framerate and drop rates). Each time this extrapolation takes place, only the most recent data provided by the monitor and the data from the benchmarks are used. In this way, a transient change in performance or network congestion does not affect future decisions. As result of the optimization process, the client might need to switch between multicast

Table 1. *Computers used in the experiments*

	Processor	Processor Speed	Memory	Graphics Card
Bachue	Pentium 4	1400 MHz	1 GB	GeForce2-32 MB
Bochica	Dual pent III	730 MHz	1 GB	FireGL 1-32 MB
Morlak	Pentium II	500 MHz	256 MB	Intense3D-16 MB

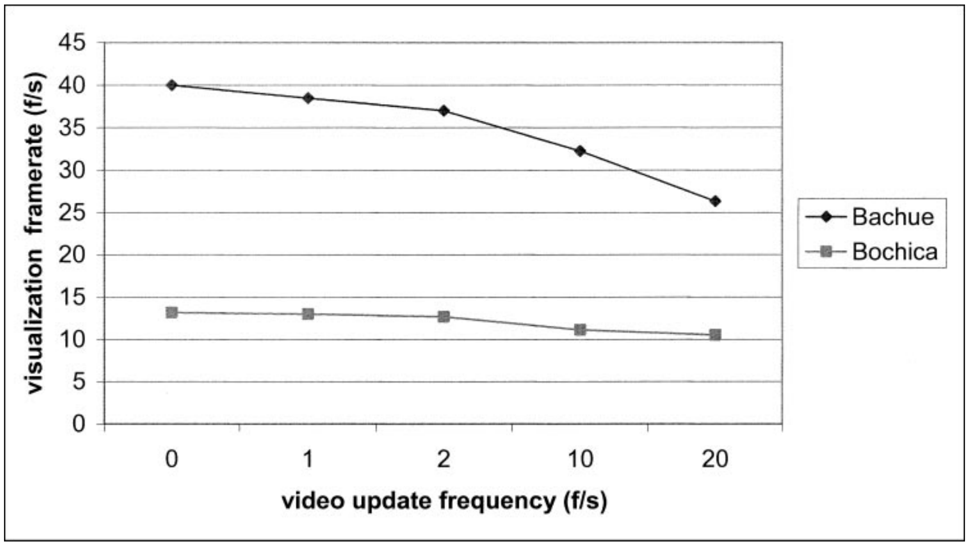

Figure 6. *Effect of video updates on the framerate in Bochica and Bachue.*

groups, which is a time-consuming operation. This limits the frequency of optimizations during the session.

4 Results

Table 1 gives the characteristics of the computers used in the experiments. The visualization data set we used for the experiments presented here is a representation of the human digestive system and consists of 27,202 vertices.

The complete performance mapping cannot be graphed because it involves five different variables. A plot of the visualization frame rate (F) versus video frame rate (V) is shown for Bachue and Bochica in figure 6. (The data for Morlak are not shown because the number of messages dropped as the video frequency increases is too high, making the data nonrepresentative.) Note the difference in the visualization framerate between the computers. Note also that, as the frequency of video messages increases, the framerate decreases noticeably. Video messages have a larger impact on the performance than other messages because they are much larger in size and require more processing.

The solutions found by the model are sensible and consistent with the data captured in the performance mapping. For instance, when the maximum priority is assigned to visualization framerate, the system lowers the graphics quality of the model to wireframe and reduces the frequency of the video updates, which negatively affect the framerate, as shown in figure 6.

The number of lost messages is an important parame-

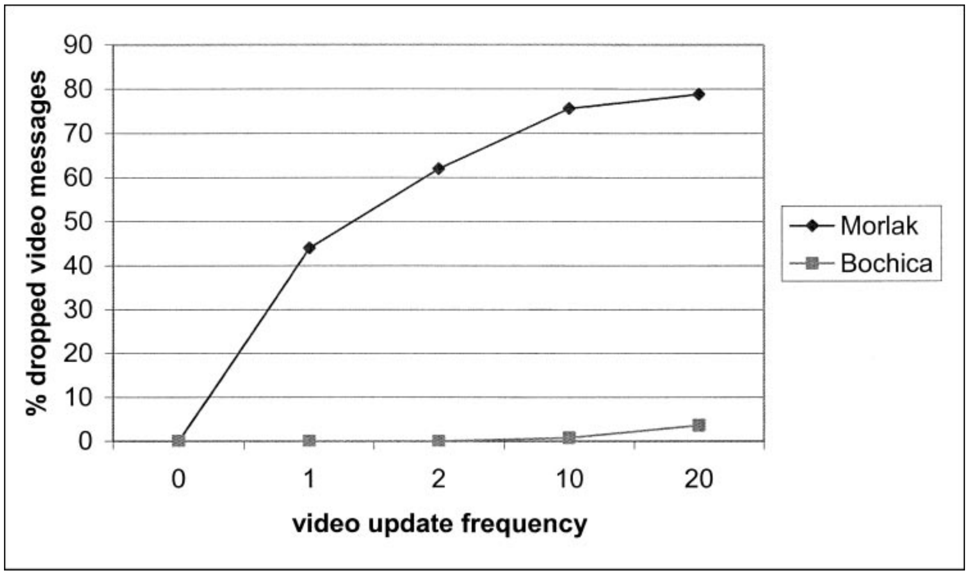

Figure 7. *Percentage of video messages dropped depending on the number of video and telepointer messages per second (in Morlak and Bochica).*

ter for measuring degradation of the system. Slower nodes drop more messages than do faster ones under the same load. Again, as the number of messages for a specific variable increases past a certain threshold, so does the number of lost messages.

Figure 7 shows how the number of lost video messages grows for Bochica and for Morlak. (The results for Bachue are very similar to those of Bochica.) Note that Bochica is capable of handling video messages without dropping any until the frequency of video frames exceeds ten per second. In Morlak, some percentage of messages is always dropped. Notice also the difference in scale: Bochica does not drop more than 6% of video messages, whereas Morlak drops up to 80%.

We have incorporated the number of dropped messages during the benchmarks into the algorithm that finds the solution. The points in the Cartesian product of variables that are found to cause the number of dropped messages to exceed the thresholds set by the administrator cannot be chosen as solutions. Those points are marked as invalid points. If the number of dropped messages is not considered, these points appear as good candidates for the solution because the frame-rate increases as messages are dropped and need not be processed. In real-life applications, different thresholds could be applied to different variables, according to the user's perception and/or relative importance.

Table 2. *Solutions for update rates of different variables found at the different nodes for the same set of user sliders' values*

	Telepointer	Object	Video	Graphics
Bachue	20	20	1	1
Bochica	20	20	2	1
Morlak	1	2	1	1

Because the mathematical model is based on actual data collected through the benchmarks, the solutions found for the same set of slider values (directly related to the objective function coefficients) vary considerably across nodes. Table 2 shows the solutions found at the different nodes for the same set of user's slider values. The user's sliders were set to $T = 4$, $O = 4$, $V = 4$, $G =$

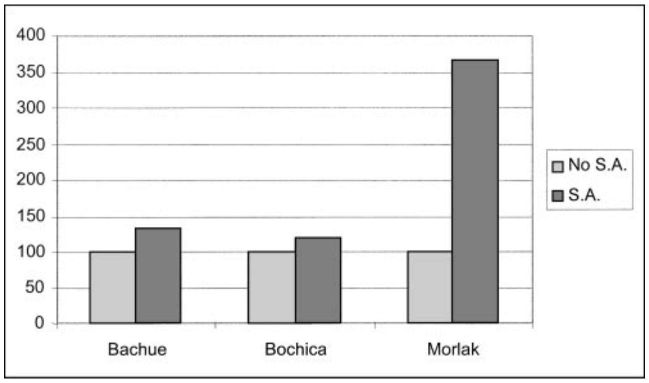

Figure 8. *The effect of employing the switchboard architecture on the visualization frame rate. Vertical axis represents the relative percentage of framerate increase.*

51, and $F = 100$. These values assign the highest importance to framerate, some importance to the graphical representation of the model, and very little to the remote events.

Note that in both Bochica and Bachue (the faster machines) updates from the telepointers and objects can still be received at maximum speed without affecting the frame rate. Morlak, on the other hand, has to reduce the frequency of all remote updates. Note also that all nodes, fast and slow, need to reduce the frequency of video updates, but Bochica, having two processors, handles video messages slightly better than Bachue, which has a single faster processor. In the current implementation, we do not consider available bandwidth, on which video messages have a large effect. Nevertheless, more-frequent video messages affect the overall display framerate as well as the percentage of dropped messages; this is how the frequency of video updates is factored into the current model.

To test the improvement in framerate when utilizing the switchboard architecture, a node was set to simulate twenty updates per second for each variable (telepointers, object, and video). Data were collected about the visualization framerate at each node with and without the switchboard architecture, and the results are shown in figure 8.

The largest improvement is naturally obtained for the slowest machine, which benefits the most from the buffering effect of the server. The visualization framerate increases found were as follows: Bachue: 33%, Bochica 10%, and Morlak 366%.

5 Related Work

One of the most desirable conditions in an NVE is to provide all participants with a consistent and up-to-date version of the shared space. But, in the presence of imperfect communication channels, the time to replicate the shared state among participants is not zero, giving rise to the consistency-throughput tradeoff ("It is impossible to allow dynamic shared state to change frequently and guarantee that all hosts simultaneously access identical versions of that state," (Singhal & Zyda, 1999, p. 102).) A considerable amount of research in NVEs has been dedicated to alleviating the limitations resulting from this tradeoff. In the following paragraphs, we summarize some of the approaches that have been explored and are pertinent to our research. We first review the approaches based on homogeneous systems and then review related work in heterogeneous systems and explain how the switchboard architecture compares to or fits into those approaches.

One of the first issues that the designer of a NVE must consider is the architecture of the system. The architecture has a significant effect on the performance, reliability, and ease of use of the system. Many different architectures have been used in the existing NVEs, ranging from highly centralized to highly distributed. SIMNET (Calvin et al., 1993), a distributed military simulation system for use on Ethernet LANs, works on replicated virtual world databases that are homogeneous. The system offers a predefined number of object types. DIVE (Hagsand, 1996) is also built around a replicated database, but it allows interactive creation and distribution of virtual objects among participants. VISTEL (Yoshida, Tijerino, Abe, & Kishino, 1995), the Virtual Teleconferencing System, used a shared world database. The system displays 3D models of each participant in the conference, and changes in a person's facial expression are sent via messages to a central server and redistributed. Only one user can modify the database at a time. An extensive survey of NVE architectures can be found in papers by Macedonia and Zyda (1997) and Funkhouser (1996).

In previous systems, we have built for exploring collaboration in 3D environments (Trefftz & Marsic, 2000), we have used peer-to-peer architectures. In the present work, however, we decided to use a client-server architecture for the reasons outlined in subsection 2.2.

A large amount of research in NVEs has been dedicated to reducing the number of messages and the computational load needed to keep a consistent, yet possibly distributed, description of the virtual world, with the main objective being scalability. Examples include area-of-interest managers (Macedonia, Zyda, Pratt, Brutzman, & Barham, 1995) and multiple levels of detail for virtual objects (Capps, 2000). Our approach uses multiple levels of detail for different modalities to meet user's preferences under constrained resources.

Techniques for reducing the number of messages commonly make use of dead reckoning, which involves sending not only the position but also the trajectory of a moving object. If each participant can locally compute the position based on the trajectory, only changes to the trajectory need to be sent. Faisstnauer, Schmalstieg, and Purgathofer (2000) improve on this scheme by assigning different priorities to groups of messages and sending the messages with higher priority before the messages with lower priority. Priority is assigned to the objects based on how much their actual position deviates from a position computed by dead reckoning. The result is that the overall systemwide error is minimized.

In the switchboard architecture, outgoing messages are also cached at the originating node. An update period is defined dynamically for each modality as a result of the optimization process. Because one of the main objectives of our scheme is to avoid fast nodes flooding slow nodes with too many messages, caching messages at the server is natural. Caching messages on the client, on the other hand, allows the administrator to establish and enforce a maximum number of messages for each modality that the system can handle. This, in turns, allows the administrator to implement quality-of-service policies.

Maxfield, Fernando, and Dew (1998) present a homogeneous NVE in which the users can register their interests as to what information they see and manipulate during the session. They study the value of being able to drop remote messages (considered as nonessential object changes) to maintain an interactive framerate within the virtual environment. The user can decide what framerate they require, and the system then determines the amount of time available between the frame renderings and let the nonessential updates use this time.

In their model, resource allocation is solved for the case of framerate and remote updates and it does not incorporate user interests in the allocation process. Our model addresses the heterogeneous case with nodes having varying resources. We formulate an optimization model that includes multiple modalities and user interests, and produces the optimal resource allocation under the given constraints.

Being able to manage the quality of service (QoS) provided to the participants is a desirable feature. Greenhalgh, Benford, and Reynard (1999) propose a system that allows users to exchange video streams in a virtual environment. The scarce multicast groups are assigned to users based on the interest of the users and their mutual proximity in the virtual environment. The QoS part of their proposed architecture consists of balancing group and individual needs. The needs of a group involve, for example, deciding which streams of information to admit into a local shared network. The needs of an individual consist, for example, in choosing whether to subscribe to streams once they have been admitted. Users express their requirements using the model of mutual awareness that has been part of their MASSIVE distributed virtual environment since its beginning (Benford, Bowers, Fahlén, Greenhalgh, & Snowdon, 1995).

In the switchboard architecture, the QoS provided by the system is dynamically monitored and adjusted. User preferences are also balanced with the resources that the system can provide. Users express their preferences by manipulating the sliders in the user interface, thereby controlling the relative importance of the available modalities. At each node, the optimizer module dynamically finds a combination of representation fidelities that satisfies systemwide and local constraints specified by the administrator while maintaining the relative importance among modalities specified by the user.

A related optimization problem is fair allocation of bandwidth among multicast sessions in heterogeneous networks (Sarkar & Tassiulas, 2000; Rubenstein, Kurose, & Towsley, 2002). Due to the network heterogeneity, a single rate of transmission per session is not appropriate because it will likely either overwhelm the slow receivers or starve the fast ones. The objective is to find a maxmin fair rate allocation so that every receiver of every session gets a bandwidth commensurate with its fair share of the capacity of the path between the source and the receiver. Our work currently does not directly consider the bandwidth available at each node and this will be part of the future work.

Middleware systems have emerged in recent years to support multimedia applications in heterogeneous computing environments. Particularly related to our research are resource brokers (Nahrstedt & Smith, 1995; Nahrstedt, Xu, Wichadakul, & Li, 2001), which manage computing and communication resources to deliver adaptive and satisfactory quality of service. Most of the existing resource brokers perform the admission test which is first-come/first-service in nature. This means that some tasks will be admitted and others will be rejected just based on their order of application. Unlike this scenario, we treat a set of dependent tasks as cooperative tasks that all need to complete an activity as a group. The model presented here reconciles user preferences and global policies under heterogeneous available resources. In our approach, the available resources determine the performance space of each computing node. The user preferences and global policies delimit a subspace of that space. The algorithm then searches for the solution that maximizes the user preference objective function. The solution essentially allocates the available resources according to the user's preferences within the limits of the valid performance space. In this way, the resource reservation, enforcement, and adaptation are accomplished in a cooperative manner.

6 Conclusions and Future Work

We have presented the switchboard architecture and the underlying mathematical model, and current results show that this architecture provides an effective way of enforcing global constraints while allowing users to adjust the quality of shared information according to their preferences.

The scenarios that can benefit from the framework we presented share the following characteristics.

- Participating nodes have diverse computing and communication resources.
- The modalities of shared information can be represented with varying degrees of fidelity in space/time dimensions.
- There are not sufficient computing resources at each and every node to represent at the same time all the modalities at their maximum level of fidelity.

In such an environment, our solution allows the following to happen in a controlled manner.

- Users can choose the fidelity or the modality for information representation to meet their preferences (within globally specified constraints or policies).
- A system administrator can define global constraints regarding minimum requirements for information representation/sharing, as well as the maximum number of messages the server can process per unit of time.

The data collected in the benchmarks that are run before the collaborative sessions form a static snapshot of the system. If conditions change during the collaborative session (for instance, if the network becomes congested), the system dynamically reacts by adjusting the benchmark data based on the actual data collected at runtime. The optimization module is invoked to search for a new solution based on the adjusted data. This feature allows the system to dynamically adapt its behavior to changing conditions during the collaborative session.

The research presented here is an initial step towards optimal handling of heterogeneous resources, user interests, and global policies in networked virtual environments. There is a great amount of research already performed in other areas of computer science on load balancing, performance analysis and optimization, and quality of service in data networks that needs to be applied to NVEs. Application of work done in those areas to NVEs is how we will eventually achieve our goal of optimality. The links need to be established to these techniques to achieve the aforementioned goal, and our continuing research lies in that direction.

Acknowledgments

The research reported in this paper was part of Helmuth Trefftz's doctoral dissertation at Rutgers University and was sponsored by NSF contract no. ANI-01-23910, the Rutgers CAIP Center, and Eafit University, Medellin, Colombia. The authors are grateful to Professor Manish Parashar for his ideas and comments, and to the anonymous reviewers, whose comments helped to improve the quality of the paper.

References

Benford, S., Bowers, J., Fahlén, L. E., Greenhalgh, C., & Snowdon, D. (1995). User embodiment in collaborative virtual environments. *Proc. CHI'95 Conf. Human Factors in Computing Systems* (pp. 242–249). New York: ACM Press.

Calvin, J., Dicken, A., Gaines, B., Metzger, P., Miller, D., & Owen, D. (1993). The SIMNET virtual world architecture, *Proc. VRAIS'93: Virtual Reality Annual International Symposium,* 450–455.

Capps, M. (2000). The QUICK framework for task-specific asset prioritization in distributed virtual environments. *Proc. Virtual Reality 2000 Conference,* 143–150.

Faisstnauer, C., Schmalstieg, D., & Purgathofer, W. (2000). Priority round-robin scheduling for very large virtual environments. *Proc. Virtual Reality 2000 Conference,* 135–142.

Funkhouser, T. (1996). Network topologies for scalable multi-user virtual environments. *Proc. VRAIS'96,* 222–229.

Greenhalgh, C., Benford, S., & Reynard, G. (1999). A QoS architecture for collaborative virtual environments. *Proc. ACM Multimedia Conference,* 121–130.

Hagsand, O. (1996). Interactive multiuser VEs in the DIVE system. *IEEE Multimedia, 3*(1), 30–39.

Macedonia, M. R., & Zyda, M. (1997). A taxonomy for networked virtual environments. *IEEE Multimedia, 4*(1), 48–56.

Macedonia, M. R., Zyda, M., Pratt, D., Brutzman, D., & Barham, P. (1995). Exploiting reality with multicast groups: A network architecture for large-scale virtual environments. *Proc. VRAIS'95 Conference,* 38–45.

Maxfield, J., Fernando, T., & Dew, P. (1998). A distributed virtual environment for collaborative engineering. *Presence: Teleoperators and Virtual Environments, 7*(3), 241–261.

Nahrstedt, K., & Smith, J. M. (1995). The QoS broker. *IEEE Multimedia, 2*(1), 53–67.

Nahrstedt, K., Xu, D., Wichadakul, D., & Li, B. (2001). QoS-aware middleware for ubiquitous and heterogeneous environments. *IEEE Communications Magazine, 39*(11), 140–148.

Riabov, A., Liu, Z., Wolf, J. L., Yu, P. S., & Zhang, L. (2002). Clustering algorithms for content-based publication-subscription systems. *Proc. ICDCS 2002 Conference,* 133–142.

Rubenstein, D., Kurose, J., & Towsley, D. (2002). The impact of multicast layering on network fairness. *IEEE/ACM Transactions on Networking, 10*(2), 169–182.

Sarkar, S., & Tassiulas, L. (2000). Distributed algorithms for computation of fair rates in multirate multicast trees. *Proc. Infocom 2000 Conference,* 52–61.

Singhal, S., & Zyda, M. (1999). *Networked virtual environments: Design and implementation.* New York: Addison-Wesley.

Sowizral, H., Rushforth, K., & Deering, M. (1997). *The Java 3D API specification.* New York: Addison-Wesley.

Tanenbaum, A. (2003). *Computer networks,* 4th ed. Upper Saddle River, NJ: Prentice Hall.

Trefftz, H., & Marsic, I. (2000). Message caching for global and local resource optimization in shared virtual environments. *Proc. VRST 2000 Conference,* 97–102.

Vickers, B., Albuquerque, C., & Suda, T. (2000). Source-adaptive multilayered multicast algorithms for real-time video distribution. *IEEE/ACM Transactions on Networking, 8*(6), 720–733.

Yoshida, M., Tijerino, Y. A., Abe, S., & Kishino, F. (1995). A virtual space teleconferencing system that supports intuitive interaction for creative and cooperative work. *Proc. Symposium on Interactive 3D Graphics,* 115–122.