On Synchronized Simulation in a Distributed Virtual Environment

Marko Meister and Charles A. Wüthrich

CoGVis/MMC [1]
Faculty of Media
Bauhaus-University Weimar
D-99421 Weimar(GERMANY)
E-Mail: [marko.meister|caw]@medien.uni-weimar.de

Abstract

This paper addresses communication problems in a distributed virtual reality system. The paper presents VOODIE, a system that provides a framework for distributed virtual environments and overcomes the communication load problems by computing as much as possible at the user end. It then concentrates on the communication load generated by a shared virtual world in a general purpose network and proves that the use of intelligent objects can reduce communication in a distributed system to a minimum. It also measures the effect of end user interaction on the network load.

Keywords: distributed virtual environments, distributed systems, interaction based simulation, active objects, object behavior

1 Introduction

In recent times, many systems [Grims91, Carls93, Green95, Reitm99, Singh99] have been developed that allow many users to access a common virtual world. Such systems can be used for instance for entertainment, cooperative work at distance, or cooperative visualization.

The first systems of this type were ad hoc solutions for the specific task [Shaw93, Singh94]. In the last decade, several generic systems have been developed to support a broad range of applications [Kazma93, Kazma95, Hubbo96, Freco98]. In parallel, middleware for the communication among objects in a distributed environment like CORBA [OMG98] or DCOM [Eddon98] have been specified. These middleware solutions, however, usually do not meet the real time requirements of a Virtual Reality application. There are, however, real time implementations [Schmi99]).

With the growth of the Internet, and of course of bandwidth, it is nowadays practicable to move out distributed virtual world applications from specialized research laboratories to the end user, connected to the Internet at home. PC technology is cheap and fast enough, and fast Internet connections are getting affordable. Already nowadays, on the Internet there are shared worlds available to a community of users [Tpres, Activ].

However, the problem of such worlds relies in the huge amount of communication necessary to simulate and synchronize a virtual world shared by many users located geographically at distance. Smart mechanisms have to be found that minimize the communication among the workstations participating to the shared environment.

There are two parameters that influence the communication in a distributed system: the topology of the network and the amount of information that has to be sent across the network so that each user has the same representation of the virtual world at a given time.

In terms of topology a communication network can be structured as a client-server system. In this case the server acts as central data store and as a crossroad for communications. The advantage of such a topology is the possibility of simple synchronization between the participants. All changes take place on the central server and the server then informs all participants. Clearly the central server becomes the bottleneck of the whole system in case of a large number of participants. Ideally, system load is better balanced in a sys-

[1] Computer Graphics, Visualization / Man Machine Communication Group

tem having no central entity. Fully distributed systems with participants having equal rights therefore have the advantage of being better scalable. On the other hand, synchronization becomes inevitably more complex.

Since 1997, we are working on a system named *VOODIE* (Virtual Object Oriented Distributed Interactive Environment) that takes the fully distributed approach. In a nutshell, in VOODIE the world is replicated and simulated locally at the user end. This implies the creation of replication mechanisms capable of insuring the consistency of the world at the user end. The world itself is seen as a collection of objects which modify their own state according to preset rules or to interaction events generated by the end users.

The objects constituting the world need to be intelligent enough to avoid as much as possible transmitting the objects current state across the network, because the number of objects that have to be synchronized equals the number of users using them.

Interaction with a virtual world implies also that users are allowed to directly modify the state of objects belonging to the shared world. However, in this case user actions are unforeseeable and more communication has to take place on the net to maintain the consistence between the object replicas across the network.

This paper focuses on exactly these two last issues, and addresses communication problems in a totally distributed system. The paper concentrates on the communication load generated by a shared virtual world in a general purpose network, proves that the use of intelligent objects can reduce communication in a distributed system to a minimum, and measures the effect of end user interaction on the network load. Tests are run on a network of up to thirty workstations simulating different usage configurations.

In section 2 we describe a general system for simulation which is based on active objects and interaction rules. The distribution of such a simulation system can be done in a natural and effective way which is presented in section 3. Section 4 deals with the interactions between objects in such a simulation system. In Section 5 we present the evaluation of our ideas in practice. Finally in section 6 we point out conclusions and starting thoughts for further research.

2 Virtual worlds as a simulation system

In many applications, Virtual Reality is used to model and visualize interactive environments. Such environments usually are constituted by entities the behavior of which is either simulated by the system, or by external interactions provided by the users of the system. To provide a correct behavior of the environment, the system has to know the functions ruling the evolution of the entities in time, and the effects of user interaction on the entities. The key here is that each entity of the virtual world is in fact a distinguishable object having its behavior in time ruled deterministically by a certain function. Thus, a virtual world is basically a collection of independent active objects each of which has a function ruling its behavior. Such functions will be called behavior function of the active object.

Similarly, also user interaction, which is mediated by the input devices of the system, can be represented by avatars, i.e. special objects representing the users, the behavior of which is ruled by the external actions of the users, which are not predictable in advance.

The global behavior of the simulated environment results from only three basic types of evolution of the objects. For single isolated objects that have no interactions with their environment the deterministic function ruling their behavior is applied to determine the state of the object at the end of the time interval considered. For objects that interact with each other -in case, for example, of a collision - the state of the object is determined by the effect of the interaction rules among objects. Finally, the state of objects interacting with avatars is changed by the effect of the interaction rules with them. Here, interactions are punctual events which change the state of the involved objects.

Such a system boils down therefore to an object oriented simulation system which runs on the time axis. Note that the idea of an interaction based simulation system is not new. For example, Bryson describes an interaction driven virtual environment [Bryso91]. However, in his system, the objects are simple passive data not integrating behavior.

In the case of a system allowing access to many users, and which in general has unpredictable bandwidths available to the end user, it makes sense to decentralize as much as possible, and therefore limit network load [Urnes99]. Many objects of the environment are active objects, and the sequence of their states can be computed deterministically: therefore it can be computed locally at the user end with a minimal load in communication. Objects and all computations regarding their autonomous behavior can be replicated and distributed on the machine where the end user is located.

3 Replicating objects to achieve distribution

In general, replicating objects having a time dependent state in a distributed system is not the simplest thing to do. The biggest problem here is maintaining the consistency of the replicas across the network. Since object behavior is deterministic in most cases, and the precision of computers with respect to time is high, especially considering the fact that in a virtual world frames have to be displayed at a relatively

low frame-rate, the synchronization of active objects across the network can take place at a quite sparse pace, and therefore communication can be reduced to a minimum for active objects.

The replication method used in VOODIE is very simple, and is based on subscription. The subscription mechanism is used to replicate objects on a number of separate simulation hosts (*stations*). If a station (slave-station) wants to create a replica of an object which is currently simulated on another station (master-station), the slave-station first requests information about all existing objects on the master-station. The master-station returns a list of object-related information. Each entry of this list contains the unique name and the type of an object. With these entries the slave-station can create replicas of remote objects by instantiating local objects of the same type as the remote objects. A newly created local object then subscribes to its master-object by sending a subscription request to the object on the master-station. The master-object then sends update information to the subscribed object when it is necessary. Note that every station can act as both master and slave-station. Since the distribution works on an per-object-basis, the role a particular station might play differs from object to object.

Each station manages a set of connections to the other stations participating in the virtual environment. These connections are channels for messages. The objects on the stations use these channels to exchange messages.

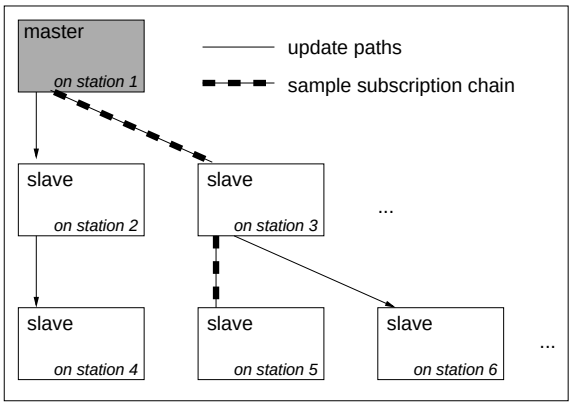

Figure 1: Sample subscription tree topology.

As the number of subscribed objects grows, it is clear that the master-object becomes overloaded. To prevent this, objects are able to forward subscription requests to already subscribed objects on other stations. This leads to a structure that we call a *subscription tree*. Figure 1 shows an example. The nodes of such a subscription tree are unique objects which reside on different stations and which are replicas of the master object.

All nodes of one subscription tree form an *object-group*. All members of such an object-group use the same behavior function, because they are instances of the same object-type. The behavior is used to calculate the state of the object at each time decentrally on each station. Furthermore the objects of an object-group are synchronized by exchanging their state from time to time so that the decentral object-state-calculation can be kept synchronized.

As an example consider an object representing time. Such a clock-object can be modeled by using a time-stamp as the state of the object. The behavior function of such an object might calculate the difference of that time-stamp and the current time on the station the object resides on. To distribute such an object we have to build a synchronized group of clock objects. For this purpose the master object is subscribed on all remote stations. This means that a subscriber object is created as a replica of the master object on a given station. The replica then requests the state of the master object. In our example this state is the time-stamp of the master object, and no more synchronization is needed.

This is only possible if all involved stations have a common timing synchronization. Zyda and Singhal pointed out that such a synchronization of time is important in networked virtual environments [Singh99]. Already in 1978, Lamport described how to synchronize time and events in a distributed system [Lampo78].

In VOODIE, we have implemented time synchronization in terms of the subscription mechanism described above. Synchronization is done similarly to NTP [Mills89], but without the need of a central time server. Time synchronization is achieved by calculating the time differences between each pair of stations. Each station keeps track of the difference between its local time and the local time of the stations to which it has an active connection. This time offset is used to correct the time-stamps of all incoming messages. Time-stamps are used to achieve simultaneous behavior of the active objects so that not only the new values are transmitted but also the time at which the change took place. The slave-objects use this information to adjust their own behavior simulation if necessary.

Note that synchronous behavior is achieved without a central entity holding a global time. The shared global time is achieved by local corrections of time between each pair of stations. Synchronous behavior is guaranteed because of the time corrections performed on the stations where slave objects reside. The measurements shown in section 5 are based on the synchronization explained above.

4 Interaction in a simulation system

Most of the time, the behavior of an object is deterministically described by its behavior function. But the be-

havior function is not sufficient to determine the object state when interaction - be it with other active objects or with avatars - occurs. As mentioned above, usually interaction occurs punctually on the time scale, and modifies punctually the state of the objects involved in the interaction.

When only objects with deterministic behaviors are involved in the interaction, there is no need to communicate the event across the subscription trees, since the object state is computed locally through the participating user stations. Every station has all the data for computing the result of the interaction locally.

A different situation occurs in the case of a non-predictable user input. Here, since the interaction is not foreseeable, it has to be communicated to all participating stations, and therefore it generates network load. In VOODIE, the interaction itself is not directly communicated (in other words, there is no special message telling the subscribers that an interaction took place). Instead, the new state of the involved objects is propagated in the subscription tree, so that the object replicas on the other stations can be updated accordingly.

Note that the number of possible interactions in such a world is as big as the power set of all master objects of the virtual environment. If only interactions between two objects are considered, however, then their number is the square of the number of master objects in the environment. For resolving interactions, VOODIE tests whether an interaction has occurred for all couples of objects.

5 Implementation and measurements

To validate the main ideas of the system presented above, we have performed tests on the VOODIE so as to analyze the network load when varying different parameters. First of all, it is tested which influence the topology of a subscription tree has on latency, i.e. delay in time in reaching a certain state between the master object and its slaves, within an object group. Secondly, the latency differences are tested when the objects are predictable active objects or avatars. Thirdly, tests are performed to measure network load by varying tree topology. Finally, network load is measured in the case of active objects or of avatars.

We have built a prototype to prove the concepts described above in practice. The simulation system, some simple active objects, interaction rules and the subscription mechanism were included in the prototype. The implementation was done in Python [Lutz96, Rossu]. Python was chosen for two reasons: first python is very appropriate for rapid prototyping and second Python allows for code-transfer across a network at runtime.

The results and measurements were produced in a heterogeneous local area network. A number of SGI workstations and Linux-PCs acted as the hardware platform for the tests. All stations were connected via Ethernet. The test environment consists of an avatar and a ball moving on a floor. Since the aim of this research is to prove that active objects reduce network load, it is not necessary to use more than one object of each type. The simulation of behavior, the interaction test and resolution, the graphical output and the communication over the network are decoupled and were executed in parallel. This is achieved by the implementation of these tasks as separate threads. Focus of the tests was to examine the synchronization between objects which were distributed over a number of stations under different circumstances.

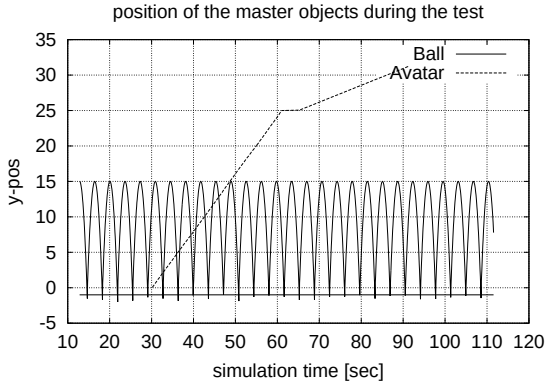

Figure 2: Position-time diagram of the master-objects used in all tests

Figure 2 shows the position time diagram for the two objects we have used for all tests. Note that the avatar object moves faster in the first part of the test. The avatar is moved 20 times per second by 0.05 units. In the second phase of the test the avatar is moved by 0.05 units 5 times per second.

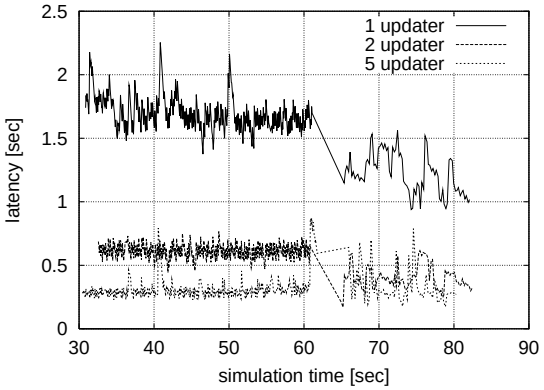

Figure 3: Latency of updates for an avatar object (27 involved stations).

The results showed in Figure 3 were achieved by performing three different tests. In all tests, an avatar-object on station 0 changes its position due to user-

input. The frequency of the change is higher in the first part of the test. These changes are unpredictable by the slave-objects. Therefore the changes must be transmitted via update-messages to the slave-objects. The number of involved stations was 27. Each line in the diagram shows the difference between the time of position-change of the master-object and the time this position-change is noticed by the last slave-object. This difference can be seen as the update-latency.

In the first test each object updates exactly one slave-object. In this case the resulting subscription tree is a chain. The length of this chain is 26.

In the second test each object has a maximum of two subscriber-objects. The resulting subscription tree has a maximum depth of four. If the objects can update a maximum of five objects, as tested in the third test, the depth of the tree is maximally two. These tests show that the latency depends on the length of the subscription chain (or update path) each message has to travel along.

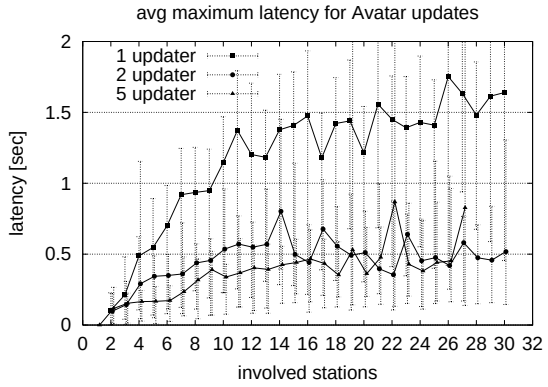

Figure 4: Average latency of updates for an avatar object for a growing number of involved stations and certain subscription tree topologies.

Figure 4 displays latency by varying the maximum number of subscribers of a single node in the subscription tree. This influences the underlying topology of the tree. The maximum number of subscribers has been set to one, two and five. The three lines in figure 4 show the average latency of avatar updates. If the number of involved Stations is S_i, a maximum of one subscriber leads to an subscription chain with a length of $S_i - 1$. If the object can update two other objects, the resulting subscription tree has a depth of approximately $log_2(S_i)$. The depth of the subscription chain is approximately $log_5(S_i)$ in the case of five subscribers. Therefore with 27 involved stations the longest path in the subscription tree is 26, 4 and 2 respectively for one, two and five subscribers. Latency depends on number of involved stations but also on the depth of the subscription tree.

Note that the network-traffic generated on a station is best balanced in the case of one subscriber. If an object has more subscribers, this object has to send more updates. This of course increases the network-traffic on the station this object resides on.

The measurements show that the latency is nearly the same with a maximum of two and five subscribers. Having two subscribers is a good compromise between update latency and network-load generated on the station.

Latency is a critical parameter of interactive systems. We also tested how the use of active objects can reduce the latency perceived by the user.

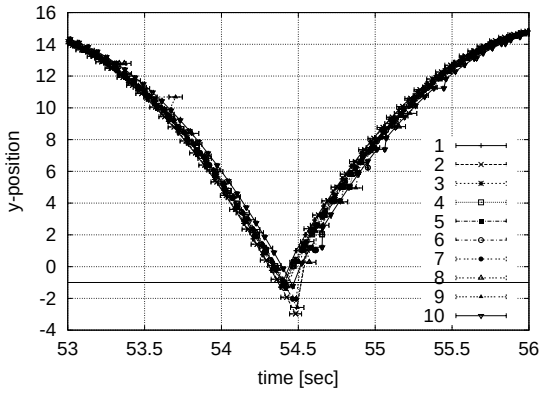

Figure 5: Position-time diagram for a group of synchronized ball objects on the different stations (10 involved stations and a maximum of one subscriber per object – the subscription tree is a single subscription chain in this case).

Figure 5 shows the traces of ball objects on different stations with respect to time. A ball object was subscribed from nine other stations. The maximum number of subscribers per object was one. So the updates were propagated from station to station. The behavior of the ball is calculated locally on every station. No update is needed as long as no interaction occurs. In the case of an interaction, some data of the ball is changed. That is, the start-position, start-speed and the time stamp of the ball. The behavior function uses these values to calculate the state of the ball. The changed values have to be transmitted to the other objects. The solid horizontal line in the diagram represents the floor. Note that the slave objects on the slave stations do the interaction resolution in parallel. Therefore the normal case should be that the state of a slave-object is already set to the state it receives via the update. The latency of an update is therefore not observable by the user.

It is notable that some object-traces lie below the floor. This results from the fact that the interactions were detected after they occur and the behavior simulation, interaction resolution, and the position measurements

are done in parallel threads. Moreover, the frequency on the time scale of these tasks are different. The small differences in time which can be observed in Figure 5 are resulting from the dynamic synchronization of time between the stations. The error could be reduced by increasing the network-traffic needed to synchronize the time.

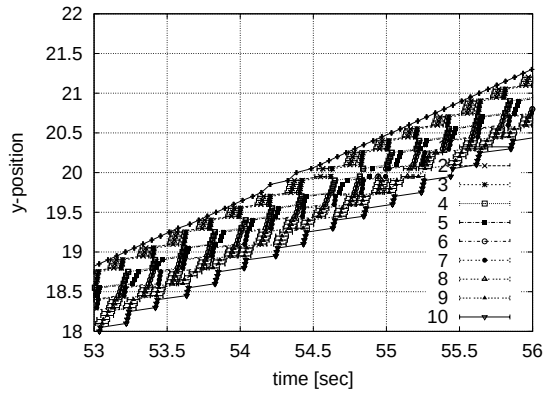

Figure 6: Position-time diagram for a group of synchronized avatar objects on the different stations (10 involved stations and a maximum of one subscriber per object – subscription topology is a chain in this case.

Figure 6 shows the traces of the avatar objects on the different stations. In contrast to Figure 5 the latencies are much higher. The reason for the higher latencies is the unpredictable character of the state-changes of the avatar-object. Since the movement of the avatar is user-based, this movement can't be pre-calculated. In this case the latency of an update is directly observable in the simulation.

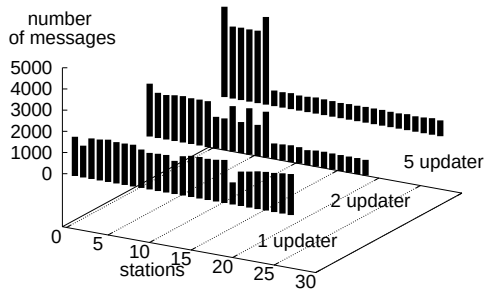

Figure 7: Amount of incoming and outgoing messages on the different stations in the case of different subscription tree topologies (27 station involved).

Figure 7 shows the number of messages each involved station has sent or received. Three test cases each with 27 involved stations were made. One can see that network load is equally distributed over all stations in the case that every object is subscribed by one subscriber. The higher the number of subscribers, the higher is the load on those stations which must update a number of other stations.

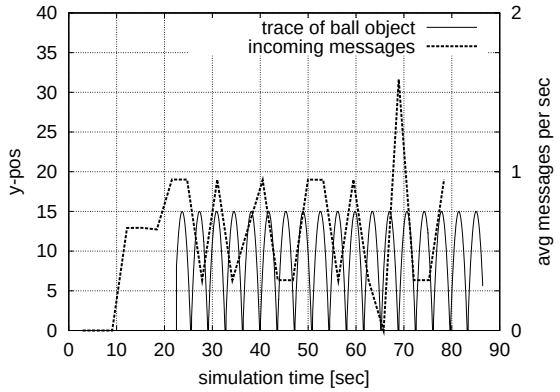

Figure 8: Position of a ball object and amount of incoming messages on a slave station.

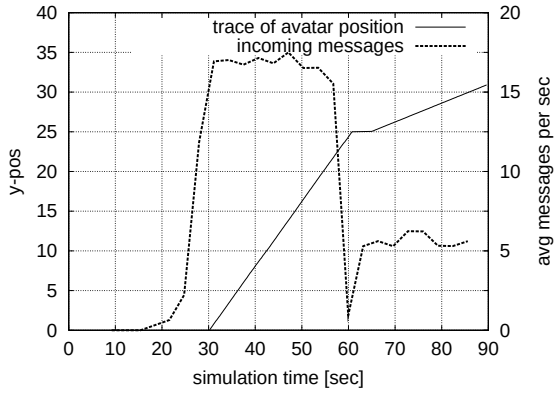

Figure 9: Position of an avatar object and amount of incoming messages on a slave station.

Figure 8 shows network load with only a ball object and Figure 9 shows network load with an Avatar object subscribed on a slave station. It is observable that the synchronization of a ball-object needs only a few messages, since the ball-object is only updated after an interaction has occurred. The avatar object needs many more updates due to the unpredictable behavior of that object. Note that the amount of messages depends on the update-frequency. Furthermore, if the avatar is not moving, no update is sent.

Globally, the use of subscription trees and of active objects proved to be a very flexible mechanism for distributing computations for a shared distributed virtual environment.

Moreover, the use of active objects considerably reduces network load, and diminishes latency in the synchronization of the objects, because the necessary calculations are made by the station at the user end. Whenever external user interaction occurs, network

load and latency increase because of the unpredictability of user input.

This is a disadvantage present also in all existing implementation philosophies. Whether using a centralized server or a distributed system, user input has to be propagated always across the system and generates network load.

6 Conclusion and future work

In this paper we presented a simulation system which is based on active objects and interaction rules. We showed that such a simulation system can be used to build a virtual environment. We presented a way to distribute and synchronize such a simulation system over a number of computers connected through a communication network.

The distribution of the simulation is based on distributing the world. It allows dynamic environments where objects can be added and removed at runtime. The subscription mechanism, presented in this paper is very flexible, due to object-based distribution and to the highly configurable topology of the subscription tree.

Our paper has tested the efficiency of the solutions adopted in balancing the usage of resources across a heterogeneous network. In particular, active objects contribute to minimize network load. Network load is one of the main bottlenecks in distributed heterogeneous systems. The solutions proposed are easily scalable, and easily adaptable to different application situations and different internal topological configurations.

Actually, this paper opened more problems than it solved. First of all, a mechanism is needed to reduce the number of interaction tests, which at this point represent the real computational problem of a distributed interactive virtual environment. Work has to be done on looking forward functions to predict interactions in advance, and to limit the tests that actually have to be done.

Second, it would be interesting to know the extendibility limits of such a system under real world usage. An attractive test environment involving real world users, such as for example a shared physically based interaction environment, should be made available. Then it will be possible to deduce the behavior of such a system and to extract consequences on the topologies implemented.

The ideas presented in this paper are relevant to a wide range of cooperative applications that go well beyond the field of virtual environments. This is in fact the major challenge opened by this paper.

Acknowledgments

None of this work would have been possible without the fruitful discussions with the members of the whole CogVis/MMC Group at the Bauhaus-University Weimar and the `[i-group]`. Special thanks are due to Jan Springer for taking the function of a permanent sparring partner during the development of this research.

References

[Activ] ActiveWorlds.com. The active worlds website. http://www.activeworlds.com.

[Bryso91] Steve Bryson. Interaction of objects in a virtual environment: a two-point paradigm. RNR Technical Report RNR-91-009, Numerical Aerodynamics Simulation Division NASA Ames Research Center, 1991.

[Carls93] C. Carlsson and O. Hagsand. DIVE - a multi-user virtual reality system. In *Proceedings of the IEEE Virtual Reality Annual International Symposium (VRAIS)*, pages 394–400, Seattle, WA., 1993.

[Eddon98] G. Eddon and H. Eddon. *Inside Distributed COM*. Microsoft Press, Seattle, WA., 1998.

[Freco98] E. Frecon and M. Stenius. Dive: A scalable network architecture for distributed virtual environments. *Distributed systems Engineering Journal(Special Issue on Distributed Virtual Environments)*, 5(3):91–100, September 1998.

[Green95] C. Greenhalgh and S. Benford. MASSIVE: a distributed virtual reality system incorporating spatial trading. In *Proceedings of the 15. International Conference on Distributed Computing*, pages 27–34, Vancouver, B.C., 1995.

[Grims91] C. Grimsdale. dVS distributed virtual environment system. In *Computer Animation, Virtual Reality, Visualisation 91*, pages 163–170. Blenheim Online, Pinner, Middlesex, 1991.

[Hubbo96] Roger J. Hubbold, Xiao Dongbo, and Simon Gibson. Maverik - the manchester virtual environment interface kernel. In *Proceedings of 3rd Eurographics Workshop on Virtual Environments*, 1996.

[Kazma93] R. Kazman. Making WAVES: On the design of architectures for low-end distributed virtual environments. In *Proceedings of the IEEE Virtual Reality Annual International*

Symposium (VRAIS), pages 443–449, Seattle, WA., 1993.

[Kazma95] R. Kazman. HIDRA: An architecture for highly dynamic physically based multi-agent simulations. *International Journal in Computer Simulation*, 5:149–164, 1995.

[Lampo78] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[Lutz96] Mark Lutz. *Programming Python - Object-Oriented Scripting*. O'Reilly & Associates, October 1996.

[Mills89] D. L. Mills. Internet time synchronization: the internet time protocol. RFC 1129, Network Working Group, 1989.

[OMG98] The Object Management Group, Inc. *The Common Object Request Broker: Architecture and Specification — Revision 2.2*, February 1998. http://www.omg.org/library/c2indx.html.

[Reitm99] G. Reitmayr, S. Carroll, A. Reitmayr, and M. Wagner. DeepMatrix– an open technology based virtual environment system. *The Visual Computer*, 15(7/8):395–412, 1999.

[Rossu] Guido van Rossum et al. The python website. http://www.python.org.

[Schmi99] D. C. Schmidt, D. L. Levine, and C. Cleeland. Architectures and patterns for developing high-performance, real-time orb endsystems. In *Advances in Computers*. 1999. to appear; http://www.cs.wustl.edu/ schmidt/RT-ORB.ps.gz.

[Shaw93] C. Shaw and M. Green. The MR toolkit peers package and experiment. In *Proceedings of the IEEE Virtual Reality Annual International Symposium (VRAIS)*, pages 463–469, Seattle, WA., 1993.

[Singh94] G. Singh, L. Serra, W. Png, and H. Ng. BrickNet: A software toolkit for network-based virtual worlds. *Presence*, 3(1):19–34, 1994.

[Singh99] Sandeep Singhal and Michael Zyda. *Networked virtual Environments: Design and Implementation*. ACM Press, Siggraph Series, New York, 1999.

[Tpres] Tpresence. The holodesk communicator website. http://www.holodesk.com.

[Urnes99] T. Urnes and T. Graham. Flexibly mapping synchronous groupware architectures to distributed implementations, 1999.