Collaboration in Persistent Virtual Reality Multiuser Interfaces: Theory and Implementation

Mohsen Kahani, H. W. Peter Beadle

Department of Electrical and Computer Engineering, University of Wollongong
Northfield Avenue, Wollongong, NSW 2522, Australia
Phone: +61-42-21-3065, FAX: +61-42-21-3236
E-mail: moka@st.elec.uow.edu.au, beadle@elec.uow.edu.au

ABSTRACT

We discuss some of the issues in collaborative persistent virtual world design and implementation for geographically-dispersed users. The development of a protocol which suits these environments are discussed. We illustrate our discussion using a persistent 3D VRML-based multiuser virtual world built for the management of telecommunication networks.

INTRODUCTION

Much research is being undertaken in Computer Supported Cooperative Work (CSCW) and Virtual Reality (VR). However little of the work reported in the literature addresses the collaboration issues of three-dimensional persistent virtual systems. A persistent system is one that exists and evolves even if there are no current users of the system. Common examples of persistent systems are network management systems and process control systems. Most persistent systems are event based and log events to a database. The events modify the state of the system model that in some way reflects the real-world system being modelled controlled. The database is also used as a data repository to provide information for users entering the system, to ensure consistency between users' views of the system, and to allow the exchange of data with other systems. Although in most systems the database is centralised, distributed databases can be used to increase system capacity if adequate synchronisation is provided.

The emergence of the Virtual Reality Modeling Language (VRML) [1] as a standard method of modelling virtual reality objects and worlds coupled to the wide spread deployment of WWW browsers allows a universal VR browser to be created. This paves the way for

participation of more geographically dispersed users in the MultiUser persistent virtual reality Interface systems (MUI).

Although Distributed Virtual Reality (DVR) games and simulation systems, such as SIMNET [2], have been around for many years, the issues related to MUI systems are different in many ways.

In most DVR system the most important objects are avatars (symbols representing other participants). The semi-static objects, such as building and bridges, are referred to as terrain, do not have a significant role in the overall system. As a result, less attention has been paid to the problems related to them. In distributed multiuser environment the situation is reversed. What is important is the semi-static objects that are manipulated by the user. The user itself, or its avatar, is not of much interest. The only important information about the user is its location, and may be its orientation, in the virtual world. This information determines the user's area of interest, so that only relevant information is presented to it. Consequently, MUIs are object-centred, as opposed to avatar-centred in most DVR systems, and require the system to be optimised for this purpose. Also, MUI systems have less tolerance for inconsistency and need more robustness. These differences lead to a set of changes in the way that information is exchanged between parties and the need for new protocol for transmission of update information.

In this paper, we discuss the issues related to persistent distributed multiuser virtual reality interfaces. In particular we focus on issues that address the collaboration between users. We firstly discuss some of the most important issues, and then explain the protocol required to support multi-user collaboration. As a case study, we will also examine a persistent VRML-based multiuser interface system for telecommunication network management.

Section I- Collaboration Issues

In this section we will discuss collaboration issues of three-dimensional persistent virtual reality multiuser interfaces. These systems should have a central location where the database is located. There are two main issues that have to be addressed: communicating data between all parties and consistency and concurrency control in the database. We will discuss these issues in the following subsections.

COMMUNICATION

The communication of updating information among the users is carried out through a network. Also, a copy of all data that is exchanged between users has to be sent to the central data repository to keep it current. Exchange of information can be done by broadcasting, multicasting or point to point communication. In broadcast communication, the data is simply sent to every site in a local area network (LAN), regardless of whether it needs, wants or should receive the data. Also, the domain of communication is limited to a particular LAN. This requires a dedicated LAN, which is usually not suitable/available for most systems. This method is used in SIMNET [2].

In multicasting method, data is sent to one or several multicast addresses, and whoever wants to receive the data joins to the appropriate multicast group. This method, while increases the efficiency of the communication, does not restrict us to a particular LAN, as well. As multicasting has not yet fully implemented in Internet, islands of multicast-capable networks are connected together using a technique called 'tunnelling', which builds a multicast backbone (MBONE) over Internet. This method has been used in newer version of SIMNET, called NPSNET [3].

Point to point communication, or unicast, establishes one connection for each participant in the system. It can be used in mesh or star architecture. In mesh model, there have to be some factors that limit the number of connection, otherwise number of connections increases too rapidly as number of users increases. This method has been used in MASSIVE project [4].

In star architecture, there is a central site to which each participant connects. Data sent to this site are resent to all other interested users. This architecture seems more appropriate for a persistent system, as it already has a central data repository. So, the central station (we call it Collaboration Manager (CM) hereafter) not only distributes the information, but also controls the database.

There are two main drawbacks for this architecture. Firstly, the CM is a single point of failure, which if it fails the whole system collapses. Secondly, the CM becomes a bottleneck both in networking and data processing if the number of users increases. Because of the need for a central database in a persistent system, the former problem exists in any other architecture, any way. For the second problem, it should be noted that in most multiuser interfaces the number of users is not too high. Also, the CM does not have much processing

task, as it mostly redistributes the data to all the other users. Nevertheless, if it happens to have dramatically effects on the performance of the system, the task of the CM can be broken into several regional computers, and some kinds of synchronisation protocol be used to maintain consistency.

Despite these drawbacks, this centralised architecture has some benefits as well. First of all, as the communication with data repository is only through one channel, the consistency and concurrency control are much simpler. Secondly, each user needs to communicate with only one computer. This will allow the establishment of a connection-oriented connection rather than a best effort one. This feature provides a reliable transport layer and lets us avoid sending transaction data repeatedly to all participants to overcome the missing packets' problem.

Area of interest

In a user interface with several independent, or semi-independent, tasks and many objects, there might be several users, each of them working on a particular part of the system. This limits the number of users interested in receiving update data for each object. In a campus-wide network management system, for example, each manager works on a particular LAN, and is not interested in receiving update data from other LANs. As a result, for each user an area of interest (AOI) can be defined and only information related to that area is broadcasted to the user.

The attributes of areas of interest, including its shape, are completely application dependent. For instance, in military simulation systems such as SIMNET, the definition and the shape of these areas are quite different from a network management application. In latter versions of SIMNET, the AOIs are hexagonal cells, with each cell associated with a multicast group. Users have the membership of several multicast groups depending on their position. As they move, they leave some groups and join some others [3].

In a network management system (as we will explain in the next section) the AOI can be defined as the current level of hierarchy that the user is within. For instance, if a user is navigating within a local area network, her AOI is all network elements and all other managers within that particular LAN segment. So if a user oversees the network from the campus level, she views each LAN as an object, and only receives update data about the overall function of each LAN, and not all the data for each individual object within LANs.

Communication between users

In a collaborative user interface, it is common that users want to communicate with each other. The communication is either via text or voice. Voice requires much more bandwidth than simple text, but provides a better interface. So, there is a trade-off between voice and text communication.

The other issue in users' communication is that to whom a message should be broadcasted. Some messages are between two parties (private conversation), and some involve more users. In the MASSIVE project [4], for instance, an aura has been defined for each user, and whenever the auras of two users collide, they can communicate with each other. With this method, the number of connection in a populated world will be limited.

It is also possible to use the concept of area of interest, and limit the audiences of conversation. Particularly in textual communication, one chat room can be allocated to each AOI, and all messages are only broadcasted to the users within the same AOI. However, the support of private conversation makes the system more complicated.

Object behaviour

Behaviour can be defined as the ever-changing state of all the attributes that an entity has. Entities can be divided into two major categories: whose behaviour is deterministic and whose behaviour is non-deterministic, each of these groups is further divided into two groups [5]. Deterministic behaviour entities are categorised as static and animated entities. Static entities never change during the simulation, such as mountains or buildings. Animated entities change over time, but their states are easily predictable. An example would be the movement of a clock's hands.

Non-deterministic entities consist of Newtonian and Intelligent entities. Newtonian entities respond to stimuli in their environment, according to the laws that their creator implemented. Intelligent entities, such as human being, have a complex behaviour, and cannot be predicted.

Considering this taxonomy, four levels of behaviour, corresponding to four types of entities, can be distinguished. In level 0, attributes are modified directly, such as "move to a specific location". In level 1, attributes change over time, such as "move at a specific speed". In level 2, a series of calls to level 1 behaviours are performed, such as "forage for food" for a bird entity. In level 3, high-level decisions are made, such as "decide whether to forage, flee,...". It is suggested to put network interface between level 1 and 2. Putting interfacing between level

0 and 1, while simplifies the simulation, places a great burden on the network. Interfacing between level 2 and 3 requires all simulator to be absolutely up to date at every frame, which is difficult to maintain. Moreover, implementation becomes very complicated.

CONCURRENCY CONTROL

Concurrency control is one of the important issues in collaborative environment. Due to the implementation complexity of an efficient concurrency control, in most CSCW system only social protocols are used. Distributing data from a central location simplifies this task, and allow us to use the modified version of available concurrency control mechanism for databases.

In management of concurrent transactions it often happens that two or more transactions have contradictory result on the final value of the database. This phenomenon is referred to as conflict serialisability in database management literature [7]. In order to manage those transactions a timestamp is attached to every transaction. Transaction scheduler in database management systems commits transaction in such an order that final value of database is the correct value. That is the one with the earliest timestamp is executed first followed by the others. This implies that the clocks of all users have to synchronised either by the CM or via any other time server.

There are several differences between data exchange in multiuser interfaces and usual transactions in databases. First of all, the transactions in user interfaces are not isolated. That means the intermediate steps are also visible by other transaction. It is also possible to interpret each task as a series of transactions. For instance, if a user grabs an object to move it to another location, all the other users see the object along its movement.

The problem is what would happen if another user wants to change the attributes of an object while the same attribute is being changed by another user. Of course, it would not be nice if, for instance, while somebody is moving an object, she suddenly realises that the object is moving in another direction, because of somebody else action. In such cases, the access of other users should be blocked until one accomplishes her task. This translates into the implementation of a locking mechanism.

The other problem is how to determine that a user has finished with a particular object. As the system is interactive, the end of each activity is not known a priori and can not be easily

predicted. As a result, an implicit or explicit lock has to be put on the object or on some of its attributes. Explicit lock means that the user explicitly locks the pieces that s/he wants to work on, and when the task is finished, those objects are explicitly released. Multiple object locking can also happen when a user locks a subtree in the hierarchy. Explicit locking is appropriate if a user wants to manipulate a set of objects, and does not want somebody else interfere until the whole task is finished.

Implicit locking should be automatically done by the system. The system keeps track of the last time each object has been accessed. So whenever it receives an access request for an object, it looks at the history, and if the same user wants to manipulate the object, the transaction is accepted. If another user wants access, the system checks the last time the object has been accessed. If the time is bigger than a specific interval, it means the previous user has finished with the object, and the access is granted. Otherwise, the access if denied and the transaction is rejected.

Whenever a transaction is rejected, the corresponding user has to be notified of rejection. Suppose that a user grabs an object to move it to a new position, if the object is implicitly or explicitly locked by another user, the transaction is rejected, and the movement should be undone. As it would be difficult that each user keeps a history for each object, and also the object may have been manipulated by the access holder in the meantime, the reject notification should include the most current position of the object, as well.

Section II- Protocol Developement

Distributed Interactive Simulation (DIS) [8] is an IEEE standard which defines different packet used for update notification between participants of a distributed simulation. However, it is based on SIMNET protocols which has been designed for VR-based military simulation. As mentioned earlier, in simulation environments the users are important, while in distributed user interface objects have greater importance than users. As a result, as DIS is optimised to be used in military simulations, it’s performance will dramatically drops if used in multiuser interfaces.

The database in DIS system is replicated. That is, each participant has a complete copy of database before it enters the simulation. All modification to the original databse are continousely broadcasted, so that newcomers can easily update their database. As a result,

there is no need for a central station to keep track of updates and modify the databse. Non of these assumptions are valid for multiuser interfaces.

We propose a star architecture with a ‘guranteed-delivery’ method for inter-user communication. User connects to the CM, and receives the most updated version of a specific view of the system from it. At the same time, an avatar representing the user, is sent to all other interested users. The shape and charactristics of area of interest is completely application-dependent. As the user navigates in the environment, its avatar moves in other users view, so that everybody can see the position of all the other users. In case a user manipulates an attribute of an object, the update notification is sent to the CM to be broadcasted to other users.

To achieve this goal, it is necessary to develop a set of Protocol Data Units (PDUs). We have tried to generalise the definition of PDUs, so that they can be used in a wider range of applications. Meantime

In this section, we try to use our discussion on the characteristics of multiuser interface environment and develop several packet data units (PDUs) that are optimised for our purpose. We firstly explain some concepts and then describe the PDUs in details.

When a user navigates within an environment, the position and orientation of its avatar constantly changes. Also, whenever s/he manipulates an objects, some attributes of that objects changes. All of these changes has to be sent to the central computer to be saved in the database and broadcasted to other users, if required. These update notifications can be transmitted using two methods: With ‘dead reckoning’, or without dead reckoning. In dead reckoning method [**Error! Reference source not found.**], the current position (or orientation) of the object is transmitted with a timestamp and a speed vector. The receiver can calculate the location of the object at any times using these information. The sender also uses dead reckoning and compares it with actual location of the object. Whenever, the calculated location is more than a threshold apart from the actual location, it again sends the new values.

Dead reckoning has been originally designed for position and orientation of objects. However, with slightly modification it can be used for other attributes, as well. For example, it is define a colour interpolator (as is define in VRML v2.0 []) and dead reckon the value of its index. Nevertheless, if an attribute does not change too much, it is not worthy of using

dead reckoning technique. Instead its absolute value can be transmitted with no further cost of extra computation.

We have designed two types of PDU for update notification: with dead reckoning and without dead reckoning. Another PDU also would be necessary if a hierarchy of collaboration managers has to be considered. The design of that PDU requires more investigation, and is not discussed here. PDUs are designed so flexible that they can be used in most multiuser interface environments. Table show the details of these PDUs.

Name	Type	Length (bytes)	Values	Description
packetID	int	1	0	
packetType	int	1	0 = received 1 = feedback 2 = rejected	
packetLength	int	2		
timeStamp	int	4		
senderID	int	2		
objectID	String	any		
attribType	int	1		
attribValue	String	any		
speedVector	String	any		
accelerator	String	any		

Table 1. Dead Reckoning PDU

Name	Type	Length (bytes)	Values	Description
packetID	int	1	1	
packetType	int	1	0 = received 1 = feedback 2 = rejected	
timeStamp	int	4		
senderID	int	2		
objectID	String	any		
attribValue	String	any		

Table 2. Non-Dead Reckoning PDU

Section III- Implementation

WWW-based Network Management System (WNMS) is a three-dimensional virtual reality multiuser interface for management of telecommunication networks [6]. This system communicates with a network management system to provide a more flexible network management system. It uses the WWW technologies such as HyperText Markup Language (HTML), Virtual Reality Modeling Language (VRML), Common Gateway Interface (CGI) script, Java and JavaScript to build an interactive distributed multiuser interface, in which the operators can manage huge networks, cooperatively. The architecture of the system is shown in

Figure 1. We will use this system as a case study to show how different issues discussed above can be implemented.

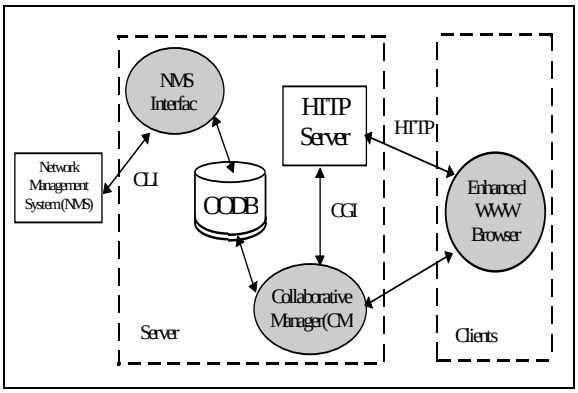

Figure 1-Architecture of WNMS

Each server consists of four parts: NMS interface, Collaborative Manager (CM), object-oriented database (OODB), and HTTP server, as shown in Figure 1. The NMS interface communicates with network management system via its command line interface (CLI). NMS can be any system capable of gathering information from network elements (NEs), and in our case is Cabeltron Spectrum. The interface queries the NMS to get management information about the status of NEs, and stores them in the OODB. It also gets update information from the database and sends them to the NMS.

The collaborative manager (CM) is the core of the system. It communicates with the clients directly, or via HTTP server, through a Common Gateway Interface (CGI) script. It also

coordinates the collaboration between clients, by collecting the update information from each client, broadcasting them to the other clients, and storing them in the OODB.

The scenario is as follows: The manager uses an WWW browser to connect to the HTTP server. After authentication, HTTP server asks the appropriate view from the CM via CGI protocol. The CM responds with the information in VRML format. The VRML script has several Java applets that firstly establish a TCP/IP connection between the client and the server, and secondly, control the behaviours of NEs in the client's environment. User, then, navigates into the 3D virtual world, interacts with NEs and manipulates the world scene. The position of the navigator and its manipulation data are continually sent to the CM via the established connection. The Java applet also listens to the connection and updates the world scene based on the receiving data.

Whenever the CM receives update data from any client, it multicasts the data to all the other clients and updates the database. If the change has to be notified to the NMS, it set a flag in the database indicating that NMS interface has to send it to the NMS. In case the user requires more information from the NMS, the request also is recorded in the database. The NMS interface, then, sends the request to the NMS, and after receiving the required data, put them in the database. The CM, finally, sends them back to the requesting client.

As mentioned earlier, the location in the network hierarchy that the user is managing was selected as the area of interests. That means that each user only receives update data about the

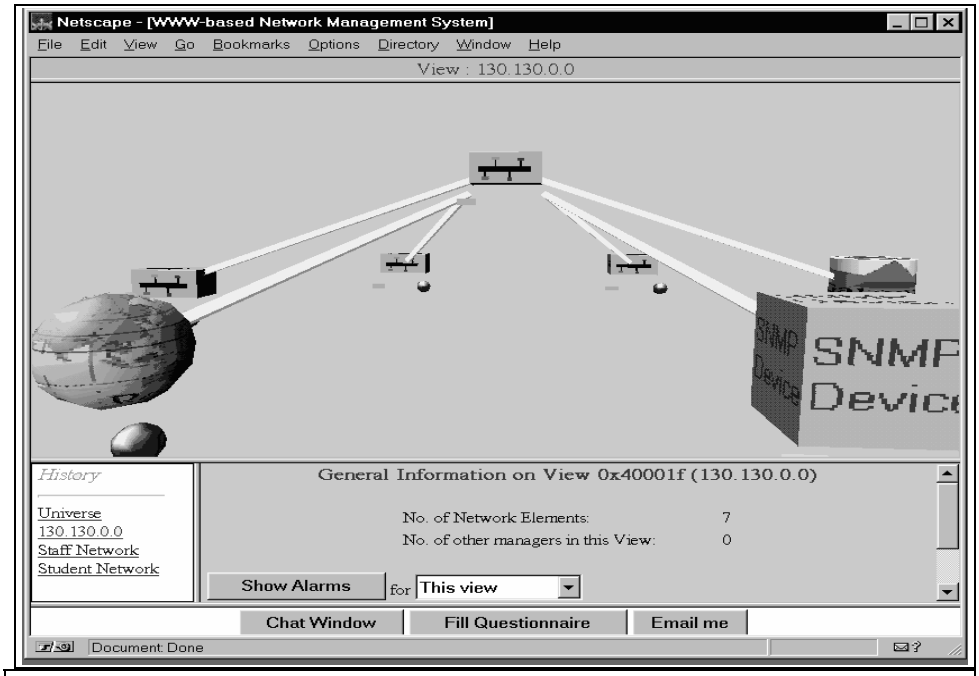

Figure 2- A typical view of WNMS

status of network elements in the same LAN segment as she is present. However, as some network elements, such as bridges and routers, do belong to more than one segment, any change in their status has to be broadcasted to all other views as well.

In order for a manager to communicate with other manager, she can open the chat window, and talk to others privately or publicly. For the moment, only a textual conversation has been implemented. Other means of communication, including voice are still being investigated. If a manager needs to attract the attention of another manager, she can click on the avatar of that user, and an alert window will open in the appropriate user's computer.

For exchange of update data, we have used the networking interface between level 1 and 2 of object behaviour. If an object is moving at a specific speed, instead of sending its position at each frame update, its location, the velocity vector and a timestamp are sent. Based on these data, the receiver can calculate the position of that object at any intermediate time. This technique is know as 'dead reckoning', and is described in Distributed Interaction Simulation (DIS) standard and related documents [8]. However, we have modified this technique, so that it can be used for other attributes of the objects as well.

This level of network interfacing has been done for other attributes of the objects as well. For instance, if a network device fails completely or partially, it starts blinking. For this situation, the only item that is sent to all users is the status of the device. Based on this information, each receiver decides that it should change the colour of the device such that it looks blinking.

To implement this system, we have developed a set of Protocol Data Units (PDUs) were developed. The design of the PDUs were based on issues discussed in the previous section. Currently, we are investigating to expand those PDUs so that they can be used in any other collaborative multiuser environments as well.

CONCLUSION

We discussed some issues related to the collaboration in persistent three-dimensional virtual reality multiuser interfaces. The issues we categorised as those related to communication of data between participants, and those related to consistency and concurrency control in the database. The most important issues in each category were discussed, and the proper solutions were presented.

We also explained, as a case study, the architecture of our VRML-based collaborative user interface for telecommunication network management. We explained how we considered the related issues and implemented them.

For the future, we are investigating on the expansion of the PDUs that we developed for the system, so that it can be used as a standard for building multiuser environment for any other context.

REFERENCES

[1] Bell, G., Marrin, C., et al., "The VRML 2.0 Specification", 4 August 1996.

[2] A. Pope, "BBN Report NO. 7102, The SIMNET Network and protocols", technical report, BBN systems and Technologies, Cambridge, MA, July 1989.

[3] M. Macedonia, M. Zyda, et al., "Exploiting Reality with Muticast Groups: A Network Architecture for Large-scale Virtual Environments", Proceeding VRAIS'95.

[4] C. Greenhalgh, S. Benford, "Virtual reality Tele-conferencing: Implementation and Experience", Fourth European Conference on Computer Supported Cooperative Work (ECSCE'95), Sweden, Sep. 1995.

[5] Bernie Roehl, "Distributed Virtual Reality: An Overview", The Proceedings of VRML'95 Conference, 1995.

[6] Kahani, M., Beadle, P., "WWW-based 3D Distributed, Collaborative Virtual Environment for Telecommunication Network Management", Accepted for ATNAC'96, Dec. 96.

[7] Papadimitriou, C., "The Serializability of Concurrent Database Update", Journal of the ACM, 26(4), pp 631-653, October 1979.

[8] ANSI/IEEE Std 1278-1993, "Standard for Information Technology, Protocols for Distributed Interactive Simulation", March 1993.