The Study of a Computer-Supported Collaborative Virtual Design System with VRML-JAVA-EAI

Hao-Ren Ke[1], Hung-Chun Chiu[2], Chien-Hung Tsao[2], Zen-Chun Shih[2]

[1]Library, National Chiao Tung University, 1001 Ta-Hsueh Rd., Hsinchu, Taiwan, R.O.C.

claven@lib.nctu.edu.tw

[2]Department of Computer and Information Science, National Chiao Tung University, 1001 Ta-Hsueh Rd., Hsinchu, Taiwan, R.O.C.

{gis87547, gis87563, zcshih}@cis.nctu.edu.tw

Abstract

This paper proposes a computer collaborative virtual design system (CSCVD) implemented by VRML, Java, and EAI. This system is a WWW-based client-server system, and messages are encapsulated in Protocol Data Unit (PDU) to transmit. PDUs are delivered via TCP. To overcome the shortcomings of TCP, this paper proposes a buffering method. The dead reckoning technology is also employed to predict the future positions of objects in order to reduce packets transferred on the network.

Keywords: Computer-Supported Collaborative Virtual Design (CSCVD), VRML, JAVA, External Authoring Interface (EAI)

1. Introduction

In fields like stage lighting, architecture, and industry design, the interaction, collaboration and communication among people (for example, designers and designers, customers and designers) are beneficial to create new ideas, reduce the time of design cycle, and design perfect products [1][6]. With the great improvement of the network speed, and the CPU power and graphical capability of computers, it is gradually becoming possible to develop systems participated by multi-users via networks; consequently, now is the right time to develop **C**omputer-**S**upported **C**ollaborative **V**irtual **D**esign (CSCVD) systems that encourage multiple users to share their ideas and participate in design processes without the limitation of location and time.

In the literature, several researches have proposed networked multi-user VR systems before; however, most of these systems are proprietary, and as a result, they are not very portable, extensible, and flexible. On the other hand, a few networked multi-user VR systems with an open architecture have also been proposed. DIVE is one representative system, which is a VRML-based system [4]. DeepMatrix [7] is another example, which is implemented with VRML, JAVA, and EAI (External Authoring Interface).

The purpose of this paper is to explore how to apply the techniques of computer graphics, virtual reality, and computer networks to the traditional collaborative design process, and present a CSCVD prototype system.

Similar to multi-user VR systems, a CSCVD system has to deal with the following three issues: (1) the rendering of virtual scenes and objects, (2) the control of virtual objects, and (3) the network communication among participants. Taking into account issues like interaction, real-time, portability, extensibility, and data sharing, we implement this system by VRML [10], JAVA, and EAI [11]. VRML is a standard for modeling 3D virtual worlds under WWW and we use VRML to render virtual scenes and objects. EAI is an interface to external application programs for controlling the local scene data of a VRML environment and we choose Java to implement EAI. As the current version of VRML does not support interaction between multiple users, we use JAVA to enable network communication among participant computers.

Our system is a WWW-based client-server system. Users taking part in a design process can connect to the system by a VRML-enabled WWW Browser. In our system, users can manipulate objects of a virtual world in many ways they want, and the manipulation of an object by a client will be seen simultaneously by all other clients. Users can also add objects into a virtual world by using a model database stored in the server or uploading objects stored locally in their desktops. In addition, they can exchange and share ideas and experiences with one another by a chat room. With the functionality incorporated in our systems, the goal of cooperative design can be achieved easily.

The organization of this paper is as follows. Section 2 describes the architecture of our CSCVD system. Section 3 focuses on the functions of our CSCVD system. Section 4 introduces the prototype system and preliminary simulation result. Section 5 gives the conclusion.

2. System Architecture

2.1. Overview

The CSCVD system proposed in this paper is a client-server VRML-JAVA-EAI system. Users participate in the system through VRML-enabled WWW browsers. Messages are transmitted by TCP. Figure 1 illustrates the interrelation among all the techniques exploited in our system [2].

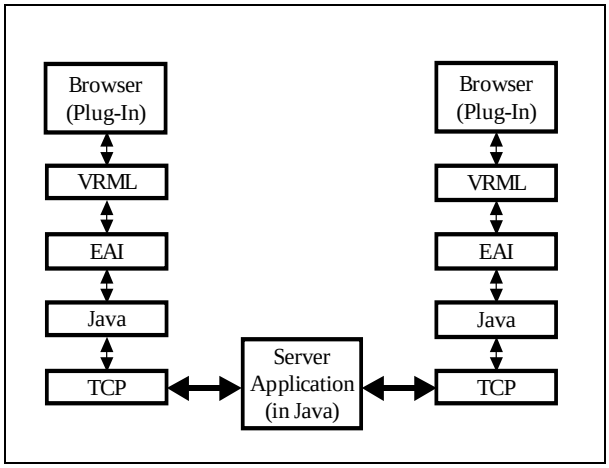

Figure 1 The interrelation among Browser, VRML, EAI, Java, and TCP

The primary tasks of a server include:

User authentication. If a user is a first-time comer, the server will request personal information from the user; otherwise, the server retrieves the user's information and usage history that is stored in a database maintained by the server. After a user logins to our system, the server will send the most up-to-date scene data to the client via HTTP.

User information maintenance. The server stores all the related information about users, including the IPs from which users connect to the server, and the avatars that are employed to represent users. In our implement, the H-anime format [5] is used to represent avatars.

Virtual scene maintenance. The server stores all the related information about a scene, objects in the scene and their associated properties. The server updates all the related information according to how users manipulate the scene data.

Message processing. The server is responsible for collecting messages sent by a client, and sending them to all other involved clients.

The primary tasks of a client include:

User interface. The user interface is responsible for accepting actions performed by users, sending messages to the server, receiving messages from the server, and invoking *EAIControl* to update the local scene.

EAIControl. EAIControl is an EAI-enabled Java class to manage the dynamic changes of a virtual world. EAI builds a bridge between a virtual world and external Java applets that manipulate it. The extensibility of our system is achieved by writing Java methods belonging to EAIControl. Most of the functions for collaborative virtual design are achieved by EAIControl.

User information and scene data maintenance. Similar to the corresponding tasks of the server.

Chat Room. The chat function is for user communication. It is responsible for sending chat messages to the server, and the server will relay messages to other clients.

2.2. Server Architecture

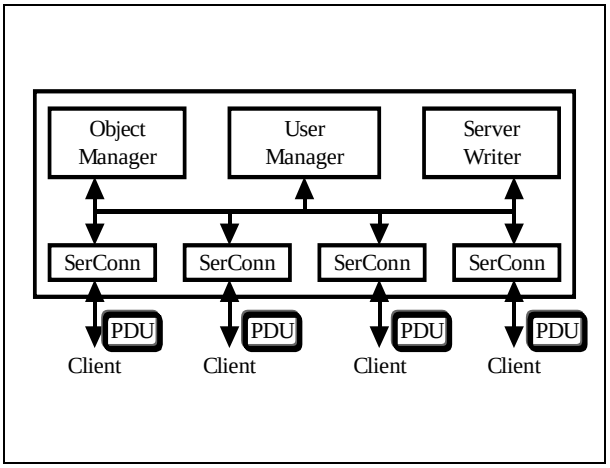

Figure 2 The server architecture

Figure 2 depicts the server architecture. The server is composed of four components: *Object Manager*, *User Manager*, *Server Writer*, and *SerConn*.

Object Manager. Our system stores virtual objects (e.g. a table, a chair, etc) in separate VRML files. From the server point of view, a virtual scene consists of instances of virtual objects, and Object Manager keeps the related information of each object instance in a scene, including an object name, the corresponding VRML file name, and its owner, position, moving direction, size, etc.

User Manager. User Manager keeps the related information of each participant, including the IP address from which the user connects to the system, user identifier, corresponding avatar model, and the object instances owned by the user.

Server Writer. Server Writer takes charge of the transmission of messages, which are encapsulated in Protocol Data Unit (PDU) [3], to clients. Our system transmits PDUs via TCP, which is reliable but slow. In order to expedite the process of message passing, PDUs are stored first in buffers managed by Server Writer, packed into a larger PDU at intervals, and then sent out. We further describe PDU and the buffering concept in Sections 2.4 and 2.5.

SerConn. SerConn is the primary control process to communicate with clients. Each time a new participant

connects to the server, a SerConn thread is created to communicate with the participant's client until the client disconnects. The tasks of a SerConn include: (1) receiving PDUs sent from clients, (2) performing necessary update according to the received PDUs, and (3) sending PDUs to the clients.

2.3. Client Architecture

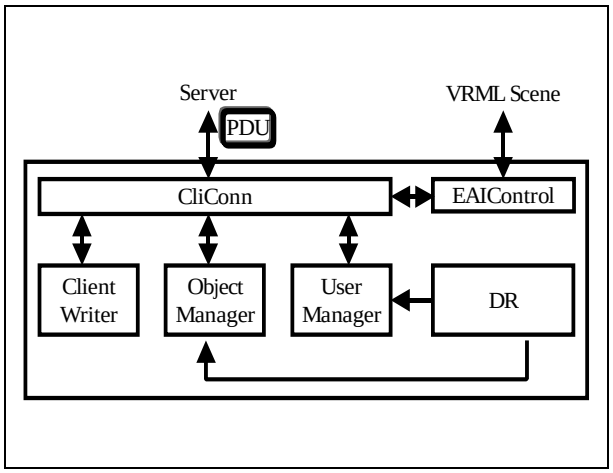

Figure 3 The client architecture

Figure 3 shows the client architecture. The client is composed of six components: *Object Manager*, *User Manager*, *Client Writer*, *CliConn*, *EAIControl*, and *DR* (Dead Reckoning [8]). The responsibilities of the client-side Object Manager, User Manager, and Client Writer are similar to their server-side counterparts, and the information stored in client-side and server-side Object/User Managers has to be synchronized. The following describes *CliConn*, *EAIControl*, and *DR*.

CliConn. CliConn is the primary control process to communicate with the server. The tasks of CliConn include: (1) sending PDUs to the server, (2) receiving PDUs from the server, (3) performing necessary update according to the actions performed by the user and/or PDUs sent by the server, and (4) calling EAIControl to update the virtual scene.

EAIControl. EAIControl facilitates virtual design. EAIControl is a Java class and consists of several methods, each of which conducts a design operation. CliConn invokes a suitable EAIControl method to conduct a design operation designated by a PDU. We further describe EAIControl in Section 3.1.

Dead Reckoning (DR). DR is an approach proposed in Distributed Interactive Simulation (DIS) [3] to forecasting the future position of an object. If the difference between the accurate and forecasting positions of an object is within a predefined threshold, no PDU for updating the object position is required to transmit; in this manner, the number of messages transmitted over the network can be reduced. In this system, we apply DR to forecast the position of objects in uniform motion and uniform acceleration motion.

2.4. Protocol Data Unit (PDU)

The concept of PDU was originally developed in DIS. As a standard packet format, PDU was used for communicating messages among distributed simulation systems. DIS proposed 6 classes and in total 27 kinds of PDUs. The PDUs designed by DIS are for military simulation and are very complicated; therefore, instead of using the original PDUs, we develop PDUs that meet the requirements of collaborative design. The PDUs proposed in the paper are divided into two classes: data transmission, flow control.

- Data Transmission: Chat PDU, File PDU, PositionUpdate PDU, OrientationUpdate PDU, AddObject PDU, DeleteObject PDU, AddAvatar PDU, DeleteAvatar PDU, and DirectionMove PDU.
- Flow Control: Login PDU, Logout PDU, Reconnect PDU, PDUPack PDU, Get PDU, and Release PDU.

Table 1 depicts the format of the *PositionUpdate PDU*. For the formats of other PDUs, please refer to [2] for details.

Content	PDU Flag	Time Stamp	Object Name	Position XYZ
Data Type	Integer	String	String	Float [3]

Table 1 The format of the *PositionUpdate* PDU

2.5. TCP Buffering

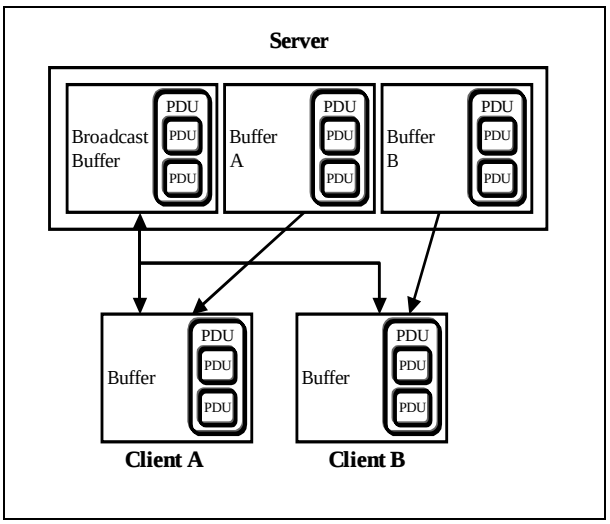

Figure 4 The buffering approach to overcome the TCP drawback

Our system uses TCP to transmit data, which is reliable but slow; furthermore, if TCP is used to transmit many small data in a very short period of time, it is neither efficient (the sender has to confirm the receipt of data) nor effective (the header of the TCP packet may be larger than the actual data). By our experiment, if TCP is used directly, on average only 30 PDUs can be

transmitted per second, which is unacceptable for a multi-user VR system. To overcome this drawback, we propose a buffering method. Instead of sending PDUs immediately, the system keeps PDUs in a buffer, packs PDUs into a *PDUPack* PDU at intervals, and then sends out the *PDUPack* PDU. This buffering approach is performed by the Server Writer and Client Writer. Figure 4 Illustrates the idea of buffering. From this figure, we can see that the Server Writer has a separate buffer for each client and has a broadcast buffer for transmitting PDUs to all clients.

2.6. System Operation Flow

In this section, we present the operation flow from the user viewpoint and system viewpoint.

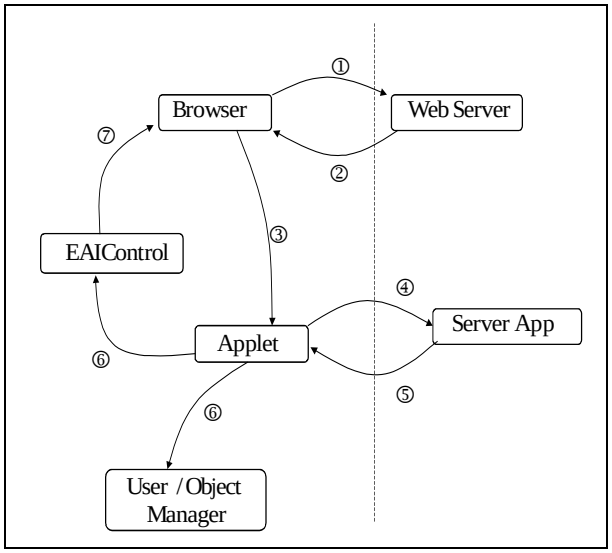

Figure 5 The operation flow from the user viewpoint

Figure 5 illustrates the operation flow from the user viewpoint:

1. A user participates in the CSCVD system by connecting to the Web server via HTTP.

2. In addition to an HTML file, the client downloads a VRML scene and the main Java Applet from the Web server.

3. The client's Browser invokes the VRML plug-in and executes the main Java Applet. The browser window shows the VRML scene, function buttons, chat room, and other menus, and the client waits for user actions.

4. The main Java Applet builds connection with the server and sends out PDUs according to user's action.

5. The server processes PDUs and sends out PDUs to those clients that should receive the PDUs.

6. The main Java Applets updates the User/Object Manager according to the PDU received, and invoke EAIControl.

7. EAIControl updates the VRML scene by invoking the EAI interface of the VRML plug-in.

8. Repeat 3-7.

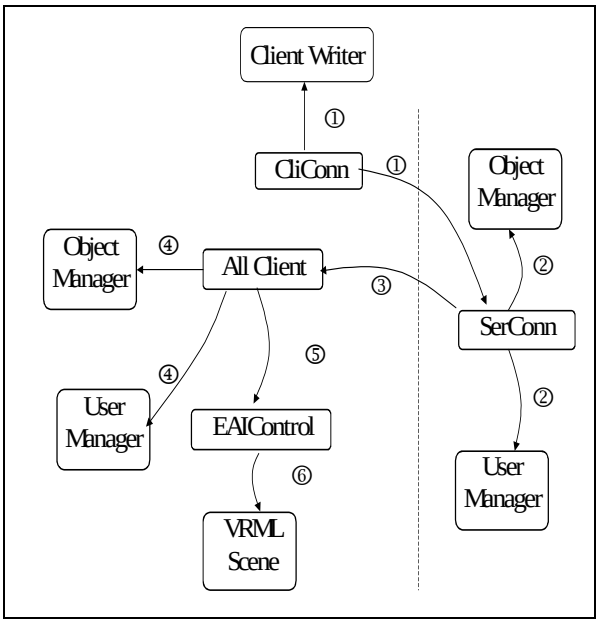

Figure 6 The operation flow from the system viewpoint

Regarding the operation flow from the system viewpoint, we take the update of an object's position as an example. See Figure 6 for illustration:

1. CliConn receives the action to move an object, creates a *PositionUpdate* PDU and stores the PDU in Client Writer. Client Writer packs the PDU along with other PDUs into a *PDUPack* PDU and sends *PDUPack* PDU to the server.

2. The server unpacks the *PDUPack* PDU and processes each PDU. When the server processes the *PositionUpdate* PDU mentioned in 1, Object Manager updates the object position.

3. The server has to inform all clients of the new position of the object; therefore, the server creates a *PositionUpdate PDU* and stores the PDU in Broadcast Buffer of Server Writer. Server Writer packs the PDU along with other PDUs into a *PDUPack* PDU and sends *PDUPack* PDU to all clients.

4. When a client receives the *PDUPack* PDU, it unpacks this PDU and processes each PDU. When the client processes the *PositionUpdate* PDU mentioned in 3, Object Manager at the client side updates the object position.

5. CliConn invokes the corresponding method in EAIControl to move the object.

6. EAIControl updates the VRML scene by moving the

object.

3. Collaborative Virtual Design

Based on the system architecture presented in the previous section, a CSCVD system is developed [9]. The purpose of this CSCVD system lies in facilitating the collaborative arrangement of 3D models and lighting, and enabling the communication among participants. In this section we briefly describe the functions of the system.

3.1. EAIControl

EAIControl is the kernel for implementing most functions supporting collaborative virtual design. Basically, EAIControl is a Java class inheriting the EAI's EventOutObserver interface. EAIControl is a collection of methods to manipulate objects or obtain the information of objects (see Figure 7). For example, we can invoke *EAIContol.getNodeTranslation(object_name)* to obtain the position of an object. Although the methods of EAIControl fulfill different functions, the underlying principle of writing methods is very similar, as illustrated in Figure 8, and we briefly describe the steps to write a method as follows. (1) Call getBrowser() to obtain the reference to a specific VRML scene. (2) Call getNode() to obtain the reference to a specific object that we want to manipulate. (3) Call getEventIn() or getEventOut() to obtain the events of an object. (4) Manipulate the object by using the events of an object.

```
public class EAIControl implements
EventOutObserver {
        public EAIControl( ) { }
        public void callback( ) { }
        public float[] getNodeTranslation( ) { }
        public void setNodeTranslation( ) { }
        public void addNewObjectNode( ) { }
        public void removeObjectNode( ) { }
        ……
}
```

Figure 7 The specification of EAIControl

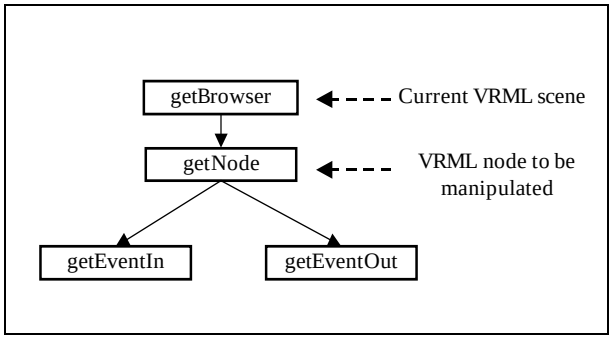

Figure 8 The naive steps to write an EAIControl method

However, in our implementation, we find that if we follow the aforementioned steps to write an EAIControl method, the performance is unsatisfactory because the references to the browser, objects, and events have to be obtained repeatedly on the fly. To improve the performance, we store the references to the EventIn and EventOut of an object into an array when the object is added into the virtual scene. While an EAIControl method is invoked to manipulate an object, it retrieves the object's references directly from the array. Figure 9 shows the modified steps to write EAIControl methods. To further enhance the performance, a hash table is employed to retrieve the reference arrays of all objects.

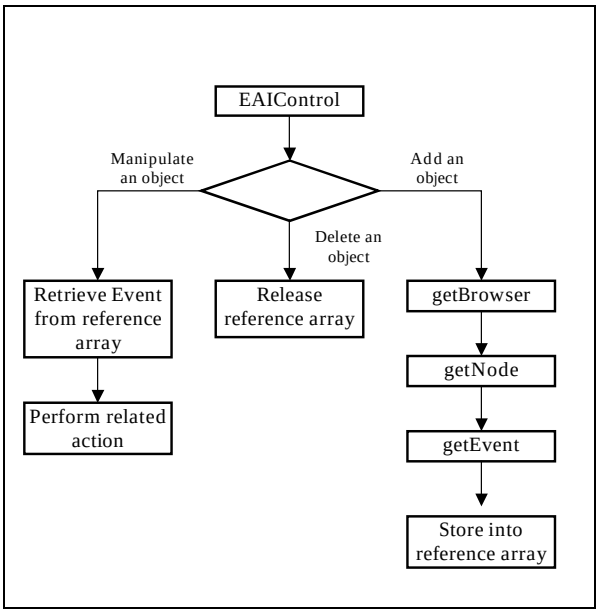

Figure 9 The improved steps to write an EAIControl method

3.2. Virtual Design Functions

Model Database. A model database stores objects that can be used by participants. The manager of our system can pre-load well-designed models into the database. In addition, users can upload models from their desktops, and in this way participants can share models they design. A Java Applet is employed to read a VRML model from a client, encapsulate the VRML model into a *File PDU*, and send the PDU to the server.

Object Selection. Before a participant performs any actions on an object, he/she has to select the specific object. We accomplish the selection of objects by using VRML's TouchSensor attached on geometry nodes and EAI's listening mechanism.

Basic Transformation. We design three kinds of interface for users to control the translation, rotation, and scaling of objects. First, users can input the precise values to control the transformation of an object; second, users can click function buttons to control the variation of an object from its current status; third, users can control the transformation by dragging an object.

Object Dragging. It's convenient for users to manipulate an object by dragging. We achieve the dragging of

objects by using VRML's PlaneSensor, SphereSensor and CylinderSensor.

Lighting Control. Our system provides users a mechanism to control the lighting of a scene. Users can manipulate spot and point light sources (by VRML's SpotLight and PointLight), fog effects, and viewpoints.

Advanced functions. In addition to the above basic functions, we devise a few advanced functions to facilitate the manipulation of objects. For example, the well known Copy and Paste functions are convenient for users to duplicate objects; the Group and Ungroup functions can treat many objects as a whole and manipulate them uniformly.

Chat Room. In a CSCVD system, communication among participants is very important. In general, communication channels can include image, video, audio, text, and among others. We implement a chat room in our system to transmit text messages among participants. *Chat* PDU is employed to carry chat messages.

3.3. Scene Loading and Storing

Usually, several runs are necessary for finalizing a design. Therefore, it is essential that a CSCVD system incorporates a mechanism to store a draft design for follow-up modification. Due to a few restrictions of VRML and EAI, we propose a method for loading and storing scenes. Basically, we have a non-VRML definition file to store the information of a scene, which is created by using the information stored in Object Manager. While a user wants to modify a previous scene, the system gives the user an empty VRML scene and adds nodes dynamically into this empty scene according to the corresponding definition file. This approach has three advantages: (1) it is easy to implement and maintain; (2) because the server has stored already the VRML files of objects, the non-VRML definition file for a scene only needs to store objects' VRML file names and coordinates, which results in a small file size; (3) by this approach, users can control every object in a scene, which is very difficult to accomplish by storing the whole scene as a VRML file.

4. Results

We have implemented a prototype system by using the ideas proposed in this paper. The server side can be executed on any machine that has installed a Web Server and Java Runtime Environment. The client side can be executed on any machine that has installed a Java-enabled WWW browser and VRML 2.0 Plug-in.

Figure 10 shows the current appearance of the CSCVD server. The upper-left part is the function buttons to start/stop the server, remove users, and add/remove objects; the middle-left window displays the actions performed by users and chat messages among participants; the lower-left part is the function buttons to move the position of objects; the upper-right and lower-right windows show the objects and users in a virtual scene, respectively.

Figure 10 The CSCVD Server

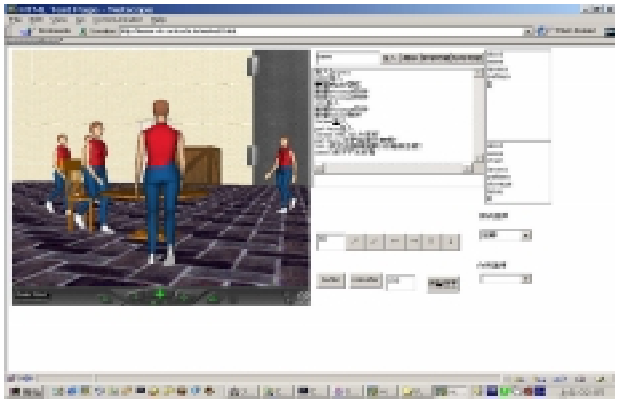
Figure 11 The Client of the first participant

Figure 11 and Figure 12 are the snapshots of the virtual scene from two participants' viewpoints. The left windows is the main window to show a virtual world; the right part is a chat window for users to share ideas and function buttons from which users can issue manipulation on scene objects, upload object models to the virtual world, identify other participants, etc. User can also manipulate objects directly via a mouse and keyboard.

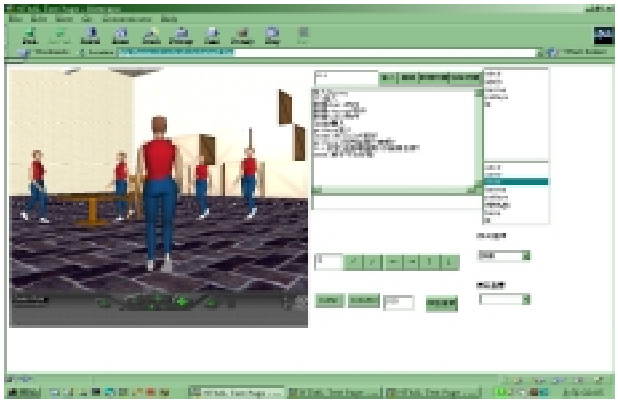
Figure 12 The Client of the second participant

To evaluate the improvement in TCP by using the buffering method, we performed a preliminary simulation. 3000 *PositionUpdate* PDUs were sent from

the server to a client. Two simulation tests were performed, one when the server and the client were connected by a LAN, and the other when they were connected by 56K modems. Table 2 shows the comparison of buffering and non-buffering method. We can see from this table that under the 56K modem and LAN networking environment, 10.19 times and 21.99 times improvement were obtained, respectively. In the future, we will perform detailed performance measure.

	Buffer	No Buffer	Improvement
56K Modem	10.015 Sec	102.19 Sec	10.19
	299.6 PDU/Sec	29.4 PDU/Sec	
LAN	4.54 Sec	99.82 Sec	21.99
	660.8 PDU/Sec	30.1 PDU/Sec	

Table 2 The comparison of TCP buffering/ non-buffering. 3000 *PositionUpdate* PDUs were sent.

5. Conclusion

This paper describes the implementation of a computer-supported collaborative virtual design system. This system is implemented by VRML, JAVA, and EAI. As VRML, JAVA, and EAI are open standards, our system fulfills the needs of portability, extensibility, and flexible. We refine the PDUs proposed in DIS to encapsulate messages transmitted in our system. A buffering method and dead reckoning are leveraged to overcome the drawbacks of TCP.

A prototype system has already been implemented, and more advanced functions will be incorporated into the prototype very soon. A preliminary performance measurement has been undertaken. With our system, cooperative design can be accomplished efficiently and effectively.

Acknowledgment

The authors would like to thank National Science Council for financially supporting this research under Contract No. NSC-88-2213-E-009-046.

Reference

1. Bridges and D. Charitos "On Architectural Design in Virtual Environments", *Design Studies*, 18(2): 143-154, 1997.
2. H. C. Chiu, H. R. Ke, and Z. C. Shih, “A Study on Distributed Multi-User Virtual Reality System,” Master Thesis, National Chiao Tung University, 2000.
3. DIS, “Standard for Distributed Interactive Simulation-Application Protocols”, Draft standard from Institute for Simulation and Training, University of central Florida, 1994.
4. O. Hagsand, ”Interactive Multi-user VEs in the DIVE system”, *IEEE Multimedia*, 31: 30-39, 1996.
5. The Humanoid Animation Working Group, http://ece.uwaterloo.ca/~h-anim/spec1.1
6. D. Marca and G. Bock, “Groupware: Software for Computer-Supported Cooperative Work,“ IEEE Computer Society Press, Los Alamitos, Calif., 1992.
7. G. Reitmayr, S. Carroll, A. Reitemeyer, M. G. Wagner, “DeepMatrix – An open technology based virtual environment system”, *The Visual Computer*, 15: 395-412, 1999.
8. J. Towers and J. Hines, “Equations of Motion of DIS 2.0.3 Dead Reckoning Algorithm, “ *10th DIS Workshop Proceedings*, 1995, pp.431-462.
9. C. H. Tsao, H. R. Ke, and R. C. Chang, “A Web-based Virtual Reality System for Real-Time Cooperative Design,” Master Thesis, National Chiao Tung University, 2000.
10. VRML Standard Version 2.0, http://vrml.org/VRML2.0/
11. VRML External Authoring Interface Specifications, http://vrml.org/WorkingGroups/vrml-eai/