3. The CommunityPlace System

3.1. Introduction

Recent trends in communications, consumer audio-visual technology, and computer devices point to a synergy creating a comprehensive electronic network that will be ubiquitous in the home, office, and the third place. Such a network will allow easy access to media and data from a variety of sources and deliver this information to users wherever they may be. To a limited extent, the recent growth of the WWW has already initiated this process.

We believe that this ubiquitous network will also provide opportunities for far greater interaction than is currently possible in the WWW. In particular, the current WWW supports multiple users but does not support collaboration among them. While it is possible for several users to access the same piece of information in the Internet, they are unaware of each other and in most cases there is little support for interaction between them.

Our goal is to begin to explore the capabilities of existing technology to support social spaces, i.e. electronic locales where many people go to meet, interact, and organize social activities. As the first step in this investigation, we have chosen to explore the 3D spatial metaphor as a basis for a shared information and interaction space. Our choice of a 3D spatial metaphor is based upon our believe that such a metaphor is an attractive "natural" environment within which users can interact. Also, 3D virtual space allows many people to share the same "place" in 3D way and have shared 3D experiences through the place. Rather than strive to find new metaphors to present data, we mimic the real world in which we live. While it is clear that not all interaction needs or benefits from a three dimensional setting, we

believe that such a setting, providing support for notions such as presence, location, identity, and activity, will provide a generic basis on which a number of different application types will be constructed.

Thus, our goal has been to build a software infrastructure that will allow many users to participate in a 3D virtual space. Their interaction will include the ability to see each other, talk to each other, visit locales, and work with each other. Our system, CommunityPlace[1] contains elements of a CSCW environment and a VR system. Such systems have already been explored in a number of experimental research platforms. However in the majority of cases the work has been confined to high bandwidth communication networks supporting small numbers of users. Our work differs in that our initial goal has been large-scale systems capable of supporting many geographically dispersed users, interconnected through low bandwidth, high latency communication links i.e. the Internet.

In this chapter, we describe the design and development of the framework for building 3D MUSVEs on the Internet. We present a platform system called CommunityPlace based upon the framework. Then we describe how the system and its multi-user application models work. We also discuss its scalability and show its performance evaluations.

3.2. Architecture for MUSVEs

Building a simple MUSVE is not difficult, even with a 3D representation. It basically requires three conceptual components: a set of devices that display a virtual environment (e.g. world), a database of objects that exist in the environment, and a set of communication links to populate the database.

Users can access the virtual environment through the display device. By using a user interface provided by the device, users can navigate the environment and interact with other users and objects in the environment. In addition to displaying the virtual environment, the display device also acts as an input device. User input, such as interactions with objects, is stored through the display device into the database and is retrieved from other display devices via a network. To achieve this, there must be some form of communication in the system that would allow display devices to access and store user input into the database. The database manages users and objects in the virtual environment and controls interaction between users and objects (or among users). Consequently, the content of the

[1] Originally, we named the system "CyberPassage". The name has been changed to "CommunityPlace" with its productizarion.

updated database is dynamically propagated to each display device via a network. These components have the following features:

- Display devices: can range from a low-end consumer electronics device, such as PDAs (Personal Digital Assistant), cellar phones, or home personal computers, to high-end graphics computers with 3D graphics hardware acceleration, such as a CAVE system (Cruz-Neira, 1993).
- Communication links: are essential for the performance of the system. Communication links of commercially available devices can transmit data at a maximum rate of less than 64 kbits per second; in contrast, a modern research laboratory has access Gbit communication links.
- Database: maintains data about the objects that make up the virtual environment and users navigating through those objects. The database is updated based upon the information from the display devices. The server (or client in case of peer-to-peer system) delivers the contents of the database to the display devices as and when needed.

There are two other significant problems that must be addressed when building MUSVEs. The first concerns the physical model used to structure the system, i.e. where the components should be placed. The second is how those components are used to support distributed algorithms and consistency guarantees.

Since the database is shared by all users' display devices accessing the database and they are updating the data, the principal role of the database is to maintain a consistent copy of the data. Changes originating at the user side need to be propagated to the database to be used to update the objects in the database in a consistent manner.

If our goal is to support a limited number of interacting users in a well-defined network setting, then the simple architecture outlined above would suffice. However, because we want to support hundreds of users, often with very different machines and types of communication access, we need to ensure that the architecture will scale up and down and will work in diverse environments.

In terms of the overall networking, since one of our basic requirements is wide area accessibility, we are naturally forced to use the Internet as a communications infrastructure. The Internet is a particularly harsh example of a wide area network. It offers low and variable bandwidth with no guarantees and manifests high and variable latencies. In addition, it is an extremely dynamic network where communication characteristics can change on a packet-to-packet basis.

As a starting point for our development, we assumed low-end, or minimum, capabilities for the above categories. In particular, we assumed that display devices

are home personal computers (PCs) without graphics support and the Internet access points are less than 64 kbps. These were the underlying assumptions behind the development of our system architecture, which resulted in the design of a hybrid client-server/peer-to-peer system model that we believe balances our goals with the constraints.

3.3. CommunityPlace System Architecture

CommunityPlace (hereafter referred to as simply "CP") system is a name of a software suit to realize 3D MUSVEs on the Internet. The CP system has been targeted towards low bandwidth and high latency networks, such as the Internet. To realize 3D MUSVEs over the Internet, the CP system adopts a natural extension of the WWW architecture and has elements of client-server and a peer-to-peer architecture.

The CP system basically consists of three individual software components: **(1) CP browser** is a 3D browser, **(2) CP bureau** is a multi-user server and **(3) Application Object (AO)** is application in a MUSVE. The CP system uses VRML97[1] (ISO standard; Hartman and Wernecke, 1996) to describe 3D virtual world and Java to describe its behaviors. Figure 3.1 shows basic system architecture of the CP system. In addition to the three components described above, the CP system assumes a WWW server and Web browser.

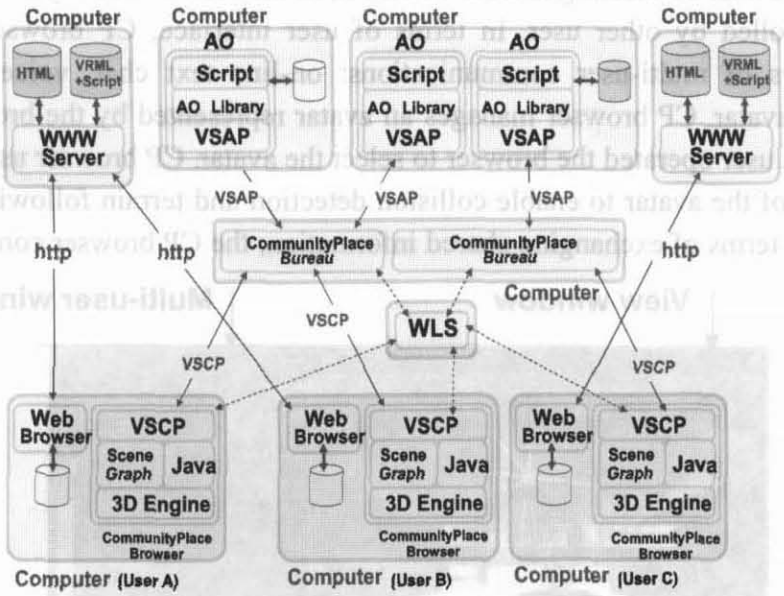

Figure 3.1 CommunityPlace system architecture

[1] http://www.web3d.org/Specifications/VRML97/. See also Appendix C.

In the following sections, we describe these three components and how they work to realize a 3D MUSVE. Then, we describe application programming models for describing multi-user applications in detail.

3.3.1 CommunityPlace browser

CP Browser is a PC-based browser to display 3D virtual worlds and allow a user to navigate through the worlds in real time. It supports VRML97 as 3D file format to describe a 3D virtual world and Java as a scripting language to describe behaviors in the world. After retrieving 3D files described in VRML97, CP browser can visualize the 3D virtual world described in the file, carry out scripts with Java system, and then allows a user to navigate and interact with the world by using a mouse and keyboard (arrow keys).

The choice of 3D scene description language has been influenced by the fact that our primary network target is the Internet and the WWW hypermedia system it supports. The predominant language used for text documents in the WWW is the Hyper-Text Markup Language (HTML). An equivalent 3D description language that compatibly works well in the WWW is VRML.

In addition to visualizing a 3D scene, CP browser provides multi-user functions for supporting MUSVE, e.g. user interface for multi-user communication, avatar management, and exchanging shared information among browsers. Figure 3.2 shows CP browser connecting to a 3D MUSVE, in the view window you can see one avatar controlled by other user. In terms of user interface, CP browser supports several ways of multi-user communications: on-line text chat, voice chat, and gestures of avatar. CP browser manages an avatar represented by the browser itself and allows a user operated the browser to select the avatar. CP browser uses physical dimensions of the avatar to enable collision detection and terrain following in a 3D MUSVE. In terms of exchanging shared information, the CP browser communicates

Figure 3.2 CommunityPlace browser

with its server, i.e. CP bureau (Section 3.3.2) and other browsers by using Virtual Society Server Client Protocol (VSCP, see Section 3.5). The CP browser detects changes to the scene geometry, which were generated by user's operations (e.g. operations to the avatar) and sends them to the CP bureau, and receives updates about the scene sent from the CP bureau to visualize them.

User interface

In default, CP browser consists of two windows: a view window and a multi-user window (Figure 3.2). See Appendix A for detail about the user interface of CP browser.

The view window displays a 3D virtual world and objects in the world. A user can navigate and interact with the world and object directly by using a mouse through the view window. Dragging the mouse on the window allows the user to navigate the world and clicking an object with a TouchSensor[1] allows her[2] to carry out its script. Several function buttons are available around the view window.

The multi-user window is available only for MUSVEs. It is popped up automatically after accessing the environment. It provides a user interface for communication and information for the environment, such as chat messages carried out around users, number of users, and names of neighbors in the world. A user can communicate with other users through the window by using a keyboard or microphone.

In addition to these two default windows, the CP browser allows world developers to extend its user interface easily by using local scripting functionality (see below).

Local scripting

Due to scalability consideration (Section 3.6), CP browser supports local scripting functionality. However, when we started this research, VRML1.0 did not yet support such scripting functionality to describe behaviors of objects in a virtual world. Therefore, we needed to extend VRML1.0 to support the functionality with our behavior mechanism based upon events and scripts (Honda et al., 1995; see Appendix B). We call the extension E-VRML (Extended VRML; See Appendix B for detail). It also supports multi-user technology for realizing MUSVEs. Figure 3.3 shows an example of E-VEML using TCL/TK as a scripting language. The

---

[1] TouchSensor can detect user's mouse operation to 3D objects associated with the sensor. See Appendix C.

[2] Note: Both female and male users exist in PAW^2, however for writing brevity we write simply "her" when referring to users of either gender.

CyberPassage system (the previous version of the CP system) supported E-VRML. This scripting mechanism of E-VRML was supported by VRML2.0 and its ISO version, VRML97 (hereafter referred to as simply "VRML"). Therefore, we have moved from E-VRML to VRML 97. We use VRML97 in this thesis.

```
Separator {
        EventHandler {
            filename "change.tcl"
            eventType GRAB
            userData "red"
            function "change_color"
            scriptType TCL
        }
        Cube {} # This cube has the event handler.
}
```

Figure 3.3 An example of E-VRML

The CP browser supports VRML and Java to describe local scripts. VRML files are downloaded into the CP browser which displays them. After that, the CP browser downloads all script files associated with the VRML files. Each script is executed by sensors which detect external events such as user's operations and then the script will generate internal events to change the virtual world visualized from the files.

Obviously, since the scripts are fully functional Java code, they are not restricted to just changing the virtual world. They can, for example, dynamically generate additional VRML nodes, or locate and add existing VRML to the base scene downloaded in the original VRML file. This mechanism enables application developers to dynamically insert new 3D objects, such as houses in the existing virtual world. This may be carried out using a call to WWW server or by a request to another network computer. Further, they can also interact with other applications, for example mining data from a database which can subsequently be turned into VRML and added to the shared scene.

Local scripting also encourages flexibility about development of a content specific user interface rather than the approach in which underlying system provides all user interfaces in advance. Application developers can dynamically extend the default user interface of CP browser by using the user interface toolkit provided by the scripting language. Java supports GUI toolkit called AWT (Abstract Window Toolkit). For example, PAW^2 provides its own user interface (control panel) implemented by using Java AWT (see Figure 4.1).

In terms of development of multi-user applications, local scripting allows application developers to realize multi-level shared behaviors (Section 3.4.4) to scale the system.

Browser-to-Browser communication

CP browser communicates with other browsers using the CP bureau (Section 3.3.2) and VSCP protocol (Section 3.5). VSCP supports application specific messages. This mechanism enables application developers to send and receive application specific messages that allow CP browsers to share events and so support shared interaction with the 3D scene. For example, a local user event causes a local script to run, which in turn uses the message sending facility of the CP system to deliver the event to remote browsers sharing the scene. At the remote browser, this network event is transformed into a local event which in turn causes execution of the local script. We discuss this mechanism in more detail in Section 3.4.1.

Architecture

CP browser provides the visualization of a 3D scene and the execution mechanism of sensors, events, and scripts to carry out animation in the scene. In addition, to achieve a goal of a shared interactive 3D scene, it allows scripts to communicate events to CP bureau (Section 3.3.2) and to other browsers sharing the scene (through the CP bureau).

Figure 3.4 shows the architecture of the CP browser. It basically consists of two components: Scene Graph Manager and Script Engine.

The Scene Manager maintains internal data structures of 3D scene as tree structure called "scene graph" (Strauss and Carey, 1992). Operations on the 3D scene can be realized as modifying the tree structure or its components. The scene graph is rendered periodically by the Scene Graph Renderer to visualize the 3D scene in real time. One of the trees is treated as this browser's avatar that other users can see in

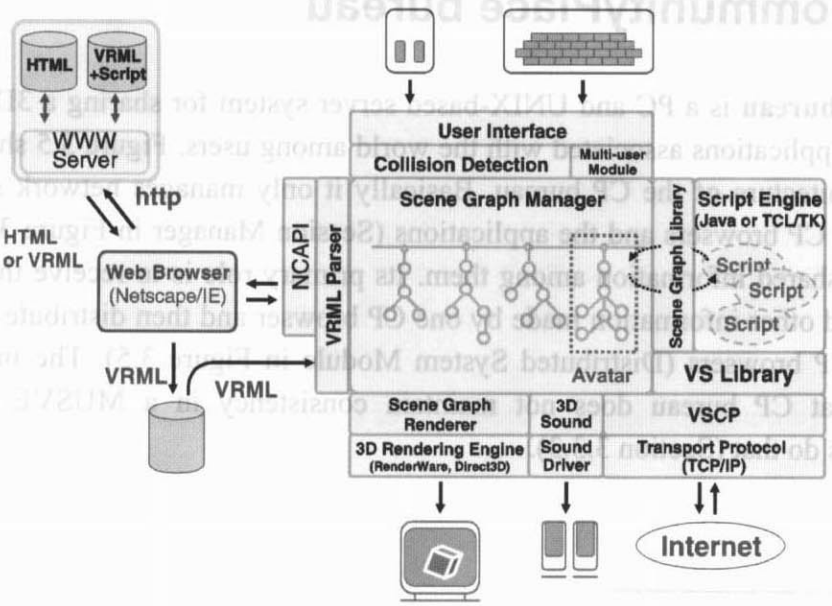

Figure 3.4 CommunityPlace browser architecture

the virtual world. A user can operate the avatar to navigate the 3D scene via the User Interface module. Collision between the avatar and the 3D scene is calculated by the Collision Detection module. When the avatar collided at something, the User Interface module provides visual feedback to the user and stops its movement. In addition, the Collision Detection module supports terrain following function to allow the user to walk on the ground or go up and down the stairs in the 3D scene. In MUSVEs, these operations on the avatar are translated into VSCP protocol through the VSCP module and sent to CP bureau via the Internet. Multi-user interactions, such as chatting, are also translated into VSCP protocol through the Multi-user Module.

The Scene Graph Manager uses NCAPI[1] (Netscape Client API) to request Web browser to retrieve VRML or HTML files. NCAPI is a communication bridge between Web browser (Netscape) and the external applications. After retrieving VRML files, VRML parser converts the files into scene graph structure and passes it to the Scene Graph Manager.

The Script Engine maintains script programs associated with the VRML scene. The VRML parser also passes script files containing the programs to the Script Engine and the engine manages the programs in its internal structure. When a user activates sensor in the VRML files, the Scene Graph Manager detects the event and passes it to the Script Engine. Then the engine executes the script program associated with the sensor. The script program can interact with 3D scene via the Scene Graph Library to change the scene and use the VS Library to communicate with CP bureau (or other CP browsers). See Appendix C about the sensor-script mechanism in VRML.

3.3.2 CommunityPlace bureau

CP bureau is a PC and UNIX-based server system for sharing a 3D virtual world and applications associated with the world among users. Figure 3.5 shows the system architecture of the CP bureau. Basically it only manages network sessions initiated by CP browsers and the applications (Session Manager in Figure 3.5), and exchanges shared information among them. Its primary role is to receive the user's position and other information made by one CP browser and then distribute them to the other CP browsers (Distributed System Module in Figure 3.5). The important point is that CP bureau does not maintain consistency in a MUSVE but the applications do that (Section 3.3.3).

---

[1] http://developer.netscape.com/support/faqs/ncapi/ncapi.html

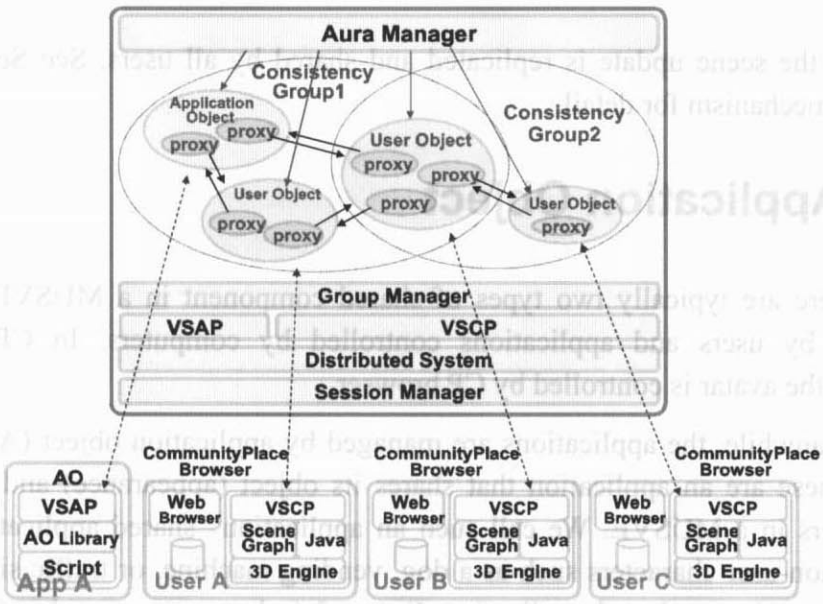

Figure 3.5 CommunityPlace bureau architecture

As described in Section 3.3.1, when a user navigates a MUSVE through CP browser, the browser sends the user's position/orientation information to CP bureau. In addition, user's interaction with the virtual world or applications generates 3D geometric changes. The CP bureau receives this information from the user's browser and distributes them to other users so that they can visualize them.

Due to scalability consideration, the CP bureau also has the role to restrict the distribution of information to CP browsers that need to know. It uses an AOI (Area Of Interest; aura) algorithm (Benford et al., 1994) to decide which other browsers need to be sent the information. An aura is the area around a user that is deemed to be interesting, anything outside the aura is not considered interesting. In CP system, all CP browsers have their aura. So the CP bureau does not send the information outside the aura of each CP browser to it. The Aura Manager (Figure 3.5) calculates collision of aura between users, between applications, or between user and application. Then it uses the results to manage consistency groups that maintain users or applications which need consistency for each other. The Group Manager supports CP bureau to manage the consistency groups. In Figure 3.5, User B and User C are in each other aura and they are in one consistency group (Consistency Group 2). This spatial technique allows us to scale the system. We describe consistency in see Section 3.6

Another role of CP bureau is to distribute message generated by local script in CP browser to other browsers in its aura. For example, user's mouse click on a shared object executes local script associated with the object. This script changes local scene graph and sends the update message to the CP bureau. Then the CP bureau distributes the message to other users to execute their local script in the same

way. Thus the scene update is replicated and shared by all users. See Section 3.4 about this mechanism for detail.

3.3.3 Application Object

There are typically two types of shared component in a MUSVE: avatars controlled by users and applications controlled by computers. In CP system, obviously, the avatar is controlled by CP browser.

Meanwhile, the applications are managed by application object (AO) in the system. These are an application that shares its object (appearance) and behavior among users in a MUSVE. We call such an application "shared application". For example, non-user characters such as a dog, vending machine, or traffic signals can be implemented as a shared application. Figure 3.6 shows an example of a shared application "dog" which can be implemented by using AO. In this example, the appearance of the dog and its behaviors, such as barking and walking are shared among the surrounding users.

AO communicates with CP bureau using Virtual Society Application Protocol (VSAP) and can work with the CP bureau in network transparent manner. Therefore, AOs can be executed on other computers which are not running the CP bureau (Figure 3.1). This architecture supports heterogeneous computer environment. For example, we can execute CPU-intensive shared applications (e.g. A.I. applications) on high performance computers not to influence the performance of CP bureau. This also provides a scalable architecture to CP system. We will describe this multi-user application programming model based upon AO in detail in Section 3.4.

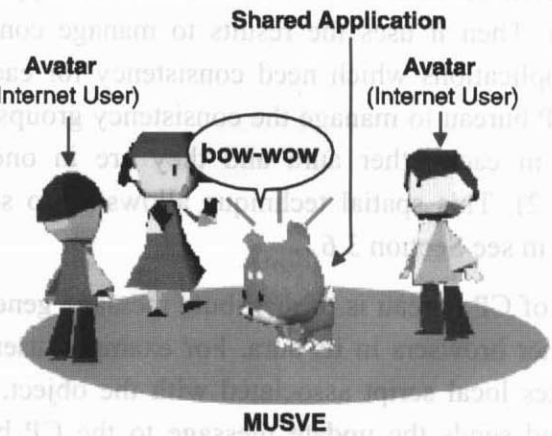

Figure 3.6 An example of shared applications

3.3.4 Relationship among components

With the Figure 3.1, we describe CP system's mechanism of sharing a virtual world. The WWW server manages VRML files which the Web browser retrieves over the Internet. Basically, the VRML files contain a description of the virtual world, such as a town. However, if the VRML file is designed as a multi-user world, it also contains multi-user information, such as an IP address and port number of the computer where CP bureau is running, and applications in the world.

(1) Suppose that a user tries to access a virtual world by using the CP system. In the WWW system, every resource can be specified by URL, including VRML. Thus, the user needs to specify the URL of the world to access the world. After the Web browser accesses the VRML file specified by the URL, the file is passed to the CP browser. Then the CP browser analyzes the file and then visualizes it.

(2) If the VRML file is designed as a multi-user world, the CP browser tries to connect to the CP bureau specified in the VRML file. This allows any browser which accesses the same VRML file to connect to the same CP bureau and so share the VRML scene. By using this approach, any VRML scene can be shared.

(3) After connecting to the CP bureau, the rest of the communication between the CP bureau and the CP browser is performed using VSCP and HTTP is used only to access files from WWW servers. The basic function of VSCP is to notify the CP bureau about any changes made by the users via their browsers and to be notified of any changes made by other browser (Section 3.5).

We will describe the scenario from user's point of view. In MUSVE, each user accessing the same environment is represented as an avatar in the environment. Thus, each user's behaviors, e.g. navigating the MUSVE are reflected by her avatar in all other browsers sharing the scene. When a user navigates around the MUSVE by moving and rotating her avatar, this movement and rotation information is sent from the CP browser to the connected CP bureau using VSCP protocol. Then the CP bureau propagates the information to all other CP browsers connecting to the CP bureau. When a CP browser receives the information from the CP bureau, it updates the corresponding user's avatar to reflect the changes made by the source browser. Applications (AO) in the MUSVE also work in the same manner.

However, this naive implementation does not allow us to scale the system. To solve the problem, CP bureau uses the AOI algorithm to limit the information sent to CP browser. We discuss this scalability issue in Section 3.6.

3.3.5 Other components

The only three components described in the previous sections are enough to realize MUSVEs over the Internet. In addition to these components, there are two other components supporting the CP system: World Location Server (WLS) and authoring environment.

World Location Server

As described in the previous section, since the CP system has a central server on the Internet, the system has an upper limit about the number of users that one CP bureau can support. In order to remove the limitation and achieve scalability, CP system supports a mechanism of multi-server system, i.e. multi-"CP bureau" system, called World Location Server (WLS). It can seamlessly extend a capacity of MUSVE. The WLS is a server system for managing multiple CP bureaus (see Figure 3.1) and has capability to switch them dynamically. Its basic strategy to switch multiple bureaus is using geographic information on virtual world. The WLS enables world developers to divide geographically a MUSVE into several sections and assign CP bureau to each section. Figure 3.7 shows how WLS works.

In Figure 3.7, suppose that a user, i.e. CP browser is in the Area1 where the Bureau1 manages and then enters into the Area2 where the Bureau2 manages. In this case, the Bureau1 detects the user's leaving from the Area1 and sends her leaving message to the CP browser (1). Then the CP browser asks the WLS about the next bureau to be connected (2). The WLS sends the information of the CP bureau, i.e. Bureau2 (3). Then the CP browser connects to the Bureau2 (4) to enter the Area2. Thus WLS can achieve seamless switching between multiple CP bureaus and easily realize one large-scale MUSVE managed by them. This mechanism is also applied to AOs when they across the boundary.

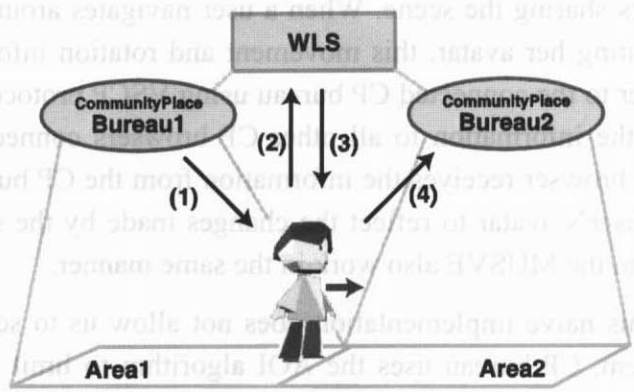

Figure 3.7 World Location Server (WLS)

Authoring environment

CP conductor (Figure 3.8) is a PC-based authoring tool for creating 3D virtual worlds. It is a kind of integration tool and does not support 3D modeling features. Rather it is used to compose scenes from 3D objects created in modeling tools to which can be added multimedia data (sound and video), textures, and behaviors. This approach allows the creation of interactive, multi-media 3D scenes or worlds and interactive execution and debugging of the behaviors.

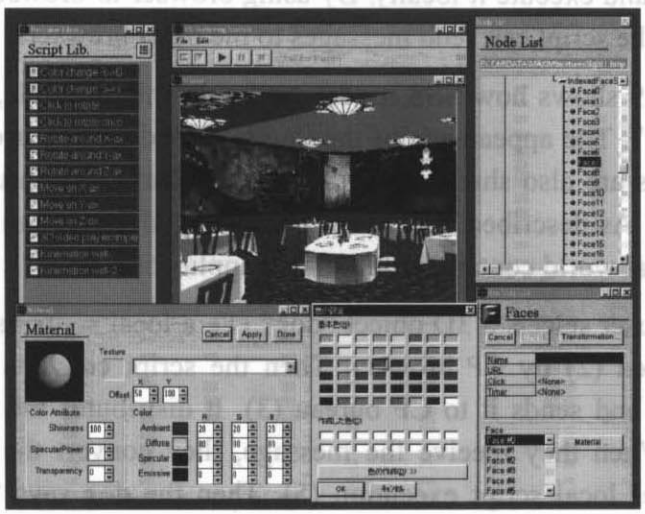

Figure 3.8 A screenshot of the CommunityPlace conductor

3.4. Multi-user Application Programming Models

CP system allows application developers to build shared applications using its multi-user application programming models. One of the important features that the multi-user application programming models should provide is to realize "write once, and share anywhere among any users". In the following sections, we will describe how the CP system realizes this feature in the programming models.

It provides two types of multi-user application programming model. The first one is called "Simple Shared Script" model and the second one is called "Application Object" model. The former is based upon a peer-to-peer model and the latter is based upon a client-server model.

3.4.1 Simple Shared Script model

The Simple Shared Script (SSS) model is a simple mechanism designed for small scale shared applications. SSS uses a replicated script model. The appearance and script of the shared application are described in a virtual world file (i.e. VRML file) using VRML's Script node (see Appendix C) and downloaded into CP browsers sharing the world. Therefore, each browser can visualize the same appearance, use the same script, and execute it locally. By using browser-to-browser communication (Section 3.3.1), the script can send messages to the other browsers.

Figure 3.9 shows how SSS model works. In this example, we use a shared application "dog". The appearance of the dog is shared among users surrounding it and its behaviors are also shared (e.g. the dog can bark when a user clicks it. See also Figure 3.6). As described above, in SSS model, all users can see the dog after loading the virtual world containing the shared application.

When a user selects a 3D object "dog" (1), a local script associated with the object is executed (2) by CP browser. Then the script converts the event into a "bark" message and sends it to CP bureau (3). It distributes the message to other browsers (4). When they receive the message, they convert the message into an event that causes local script execution (5). Then the dog says "bowwow" in all browsers. Thus, updates generated by the script are shared among users.

The advantage of the SSS model is to write a single script in VRML manner. This allows application developers to develop and maintain the shared application easily. The drawbacks of the SSS model are based upon ownership and persistence of the shared application. Since all scripts of the shared application are equal, they need to communicate among themselves to resolve their ownership and persistence.

To solve these problems, we introduced a mechanism to assign master

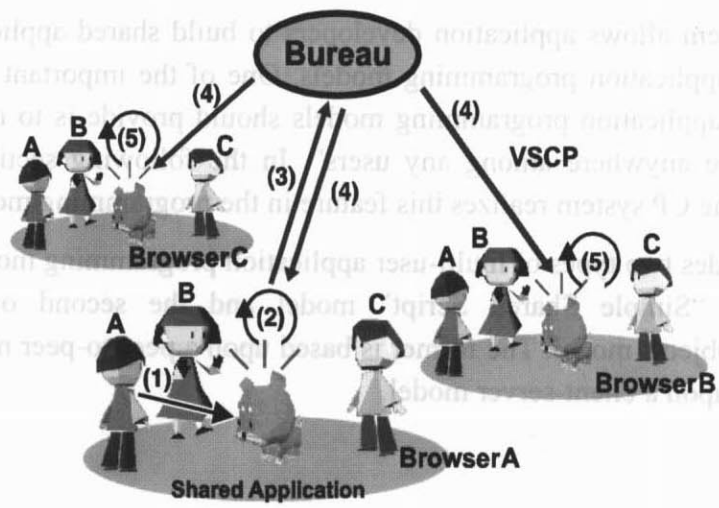

Figure 3.9 Simple Shared Script Model

ownership to one CP browser. It can receive messages from other CP browsers and send messages to them to realize a central logic in the SSS model. When the CP browser which has master ownership is terminated, CP bureau detects the event and assigns new master ownership to other CP browser. We tend to use the SSS model for simple shared applications that do not need sophisticated synchronization or persistency requirements.

3.4.2 Application Object model

While the SSS model supports simple shared applications, more complicated shared applications need more sophisticated mechanism. To realize such mechanism, CP system provides a mechanism of "application object" which can exist externally to CP browsers and CP bureau.

Application Object (AO) is the entity which is responsible for handling application specific behaviors of 3D objects in a MUSVE. It allows application developers to create 3D objects and inject them in the existing MUSVE dynamically. Thus, the application developers can easily change the appearance and behaviors of the MUSVE from outside of the environment.

Development environment of AO is provided for Java and C++. It supports the following APIs to:

- Create, delete AO,
- Set event handlers (e.g. timer event handler, event handler for either users or other AOs entering into the AO's aura, and event handler for application specific message),
- Control the AO, such as setting position and orientation of the AO,
- Access internal information in CP bureau managing the AO, and
- Exchange messages between AOs.

Three components

AO consists of three components: 3D object (appearance) file, a client-side script associated with the object, and its server-side script. Figure 3.10 shows the relationship among these components.

- **3D object file:** defines a 3D appearance of AO and its initial spatial position/orientation in a MUSVE. It is described in VRML file and downloaded into CP browser. It also binds the appearance and the following client-side script in VRML manner (i.e. using Script node).

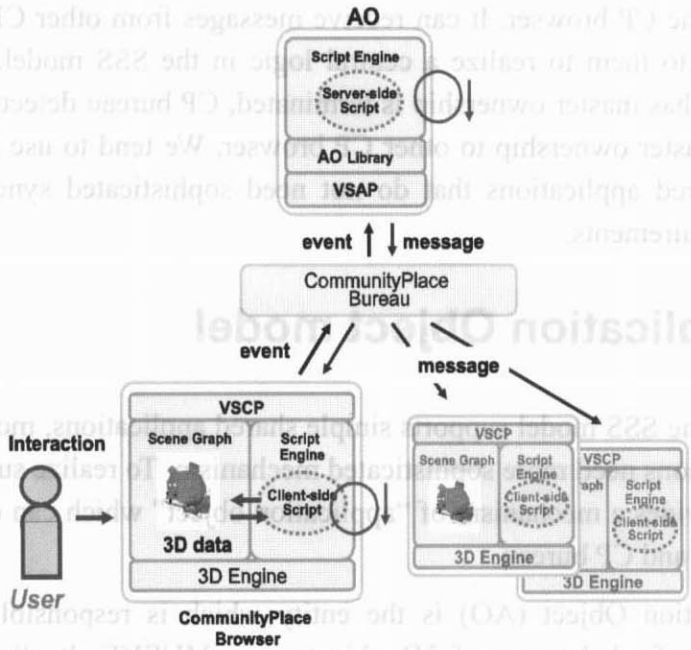

Figure 3.10 AO architecture

- **Client-side script:** controls interactions between a user and the AO by using the Scene Graph Library (Figure 3.4) and message passing between this script and the following server-side script by using the VS library (Figure 3.4). This script is associated with the 3D object in VRML manner, and equally downloaded into CP browser like other scripts. It is described in Java. For example, when a user clicks an object with sensor, the event is passed to this script through the Scene Graph Library. Then the script sends it as messages to the server-side script by using the VS library (Figure 3.4).
- **Server-side script:** controls a central application-specific logic of the AO and decides its behavior. In addition to the logic, it contains the file name (URL) of the 3D object file to bind the 3D object and this script. It also controls message passing between this script and the client-side script[1] by using AO library (Figure 3.10). It is described in Java or C++.

Relationship among components

Figure 3.11 and Figure 3.12 show how AO model works in a MUSVE. We use the same example, i.e. the shared application "dog" described in Section 3.4.1. In this case, it is implemented using AO.

[1] A single message is sent from server-side script to client-side script. As you can see in Figure 3.10, the message is sent to CommunityPlace bureau and then the CP bureau duplicates it and multicasts it to plural browsers (client-side scripts).

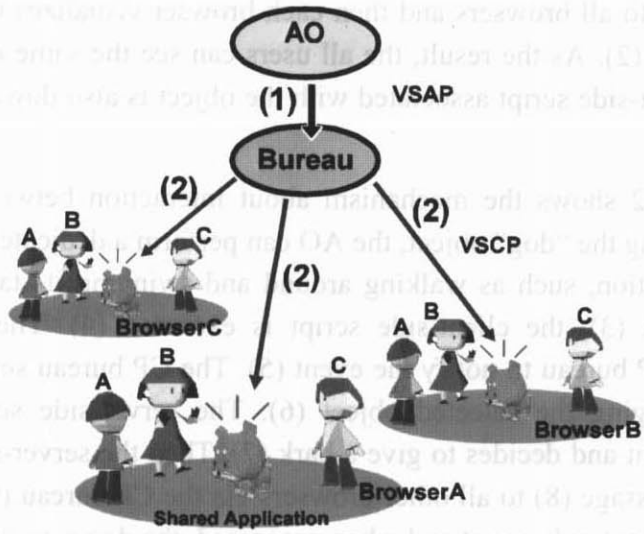

Figure 3.11 The mechanism of AO (1)

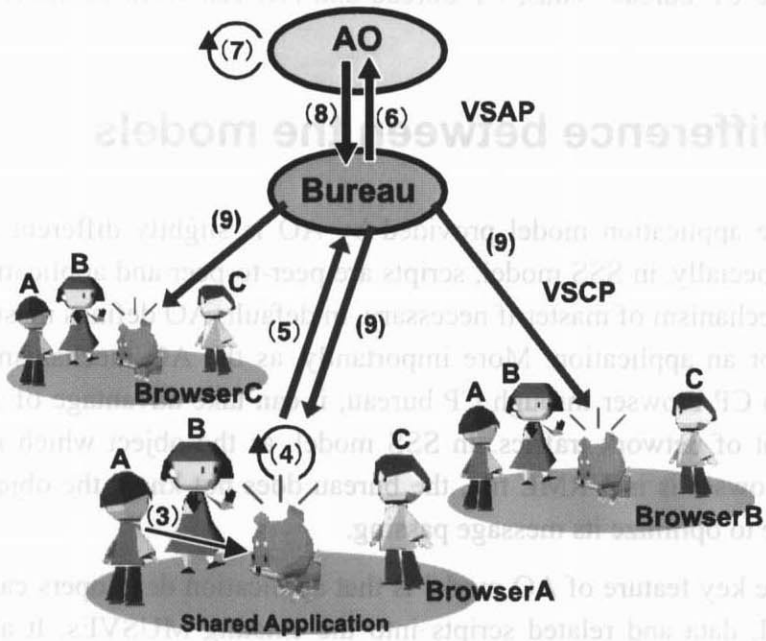

Figure 3.12 The mechanism of AO (2)

Figure 3.11 shows introduction of the AO into a MUSVE populated by three users. It means that three CP browsers are connecting to the CP bureau managing the MUSVE. When the AO is executed and attached to the CP bureau via a network, it requests the CP bureau to create and inject 3D object described in the 3D object file into the MUSVE (1). This registration informs the CP bureau about the visual

representation of the 3D object and the spatial position of the object. The CP bureau sends the request to all browsers and then each browser visualizes the object at the specified position (2). As the result, the all users can see the same object "dog". In addition, the client-side script associated with the object is also downloaded into the CP browsers[1].

Figure 3.12 shows the mechanism about interaction between the AO and user. After realizing the "dog" object, the AO can perform a dedicated processing for the "dog" application, such as walking around and swinging its tail. When a user selects the object (3), the client-side script is executed (4). The script sends a message to the CP bureau to notify the event (5). The CP bureau sends the message to the AO managing the selected object (6). The server-side script of the AO processes the event and decides to give a bark (7). Then the server-side script sends back a "bark" message (8) to all other browsers via the CP bureau (9). The message is passed to the client-side script and when processed, the dog says "bowwow."

The AO sits out of the CP bureau and communicate with it using VSAP protocol (see Figure 3.1) to register the AO, send requests, and receive messages to/from the CP bureau. Thus, CP bureau and AO can work in network transparent manner.

3.4.3 Difference between the models

The application model provided by AO is slightly different from the SSS model. Especially, in SSS model, scripts are peer-to-peer and application developers can use mechanism of master if necessary. In default, AO defines master and control of logic for an application. More importantly, as the AO mechanism registers the object into CP browser through CP bureau, it can take advantage of AOI to reduce the amount of network traffics. In SSS model, as the object which is downloaded into CP browser is in VRML file, the bureau does not know the object and so it is impossible to optimize its message passing.

The key feature of AO model is that application developers can dynamically add VRML data and related scripts into the existing MUSVEs. It allows them to build MUSVEs that evolve over time. Basic scene descriptions are described in main VRML files and CP browser downloads the files. After the CP browser carried out the MUSVE, they can make applications using AO and add the application to the basic scene. In commercial environments, thus service provider can add applications

---

[1] As the CP system is a natural extension of WWW system, these requests contain a URL to specify the related resources, such 3D objects or scripts. When the CP browser receives the request, the CP browser retrieves the 3D objects or scripts by using the URL via HTTP (Section 3.5.1).

to the existing MUSVE dynamically. For example, 3D shopping mall consists of basic 3D scene, such as shop buildings, which is initially downloaded into CP browsers. Subsequently, a service provider can add a new shop to the mall by creating its AO and connecting to the CP bureau managing the mall. This model allows separating between server (bureau) manager and service providers and providing an open and extensible mechanism for application provision.

3.4.4 Different consistency levels

In terms of consistency management, there are obviously many variations within these models. Figure 3.13 shows the variations, i.e. different levels of consistency management. It shows high, middle, and low level model. These models describe a shared application robot and its shared behavior (i.e. the robot can bow when a user clicks it). In Figure 3.13, there are three users in the robot's aura. When a user clicks the robot, the behavior will be carried out. In the following descriptions, "latecomer" means a new user who enters the aura lately.

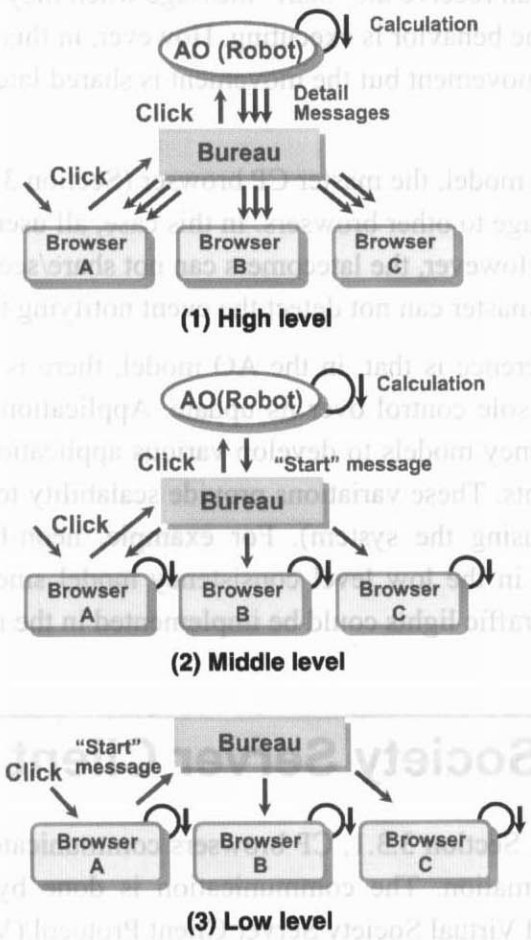

Figure 3.13 Different consistency models

- In high level model, after receiving the click event, the AO sends the detail messages describing the behavior (e.g. every absolute angle of the robot's head from its initial position, such as 5, 10, 15, 20, 25, and 30 degrees) to the AO's client-side script. The client-side script sets the angle of the head whenever the script receives the messages. In this case, all users can share/see the accurate movement of the robot. The latecomers can also share/see that when they entered the AO's aura during the behavior is executing. This is because the CP bureau/AO can detect the event notifying that they entered the aura and request their CP browsers to realize the AO. Then the latecomers can receive the message to realize the same movement accurately.
- In middle level model, the AO sends only the "start" message to its client-side script when it receives the click event. The client-side script carries out the behavior locally after receiving the message. In this case, all users can share/see the same movement. Also, the CP bureau/AO can detect the latecomers and request their browsers to realize the AO. The latecomers can receive the "start" message when they entered the AO's aura while the behavior is executing. However, in this case, they can see the "same" movement but the movement is shared later than the three users.
- In low level model, the master CP browser (Section 3.4.1) sends only the "start" message to other browsers. In this case, all users can see/share the movement. However, the latecomers can not share/see the movement because the master can not detect the event notifying them.

The major difference is that, in the AO model, there is a designated owner for an object who has sole control over its update. Application developers can use these different consistency models to develop various applications based upon their consistency requirements. These variations provide scalability to CP system (i.e. the MUSVEs developed using the system). For example, neon-light advertisements could be implemented in the low level consistency model since they do not need accurate consistency. Traffic lights could be implemented in the middle level model.

3.5. Virtual Society Server Client Protocol

As described in Section 3.3.1, CP browsers communicate with CP bureau to exchange shared information. The communication is done by using a dedicated network protocol called Virtual Society Server Client Protocol (VSCP).

The VSCP has two goals: efficient handling of 3D geometric transformation of 3D objects (i.e. translation, rotation, and scaling), and open-ended support for application extensibility (i.e. application specific messages). The first goal is achieved by ensuring that VSCP has a very compact representation of the 3D transformations and control messages. This efficiency is obviously crucial considering our target of dial-up connections. For the second goal, VSCP has an object-oriented packet definition that allows applications to extend the basic packet format with application specific messages. VSCP runs above TCP which provides connection guarantees and simple firewall traversal.

3.5.1 Object-oriented message passing for 3D objects

To work with low bandwidth communication links, we require that 3D object transformation messages should be sent with higher performance than other types of messages. Special care must be taken for user representative object movement.

We define a generic messaging system over 3D objects. That is, all 3D objects appearing in a 3D virtual world are treated as objects in the messaging system and can be targets for messages. The base message set is made up of 3D geometric operations such as translation, rotation, and scaling. Application specific message types can be defined as extensions to the base message set. The message format consists of three components: (1) a target object identifier, (2) an operator, and (3) its arguments. Each 3D geometric operation is assigned a predefined operator.

There are two exceptions to this; object creation and deletion are treated in different manner. In CP system, all the object creation is initiated by the server side as a result of application's request (Section 3.4.2). This is because the server, i.e. CP bureau needs to know the object identifiers to correctly route messages and only the CP bureau can easily guarantee that these identifiers are unique. In addition, as described in Section 3.3.2, the CP bureau needs to manage aura collision among the objects.

If CP bureau wants to create objects in any CP browser, the CP bureau sends a message "AddObject" to the CP browser. There are two arguments of the message. The first argument is an identifier for the new object allocated by the CP bureau, and the second argument is a file descriptor for a VRML file describing the 3D shape of the object. The file descriptor is either a URL or a local file name in the browser side. In either case, the file should contain VRML which describes the object's shape. Object deletion is also initiated by the CP bureau. The CP bureau sends a

"DeleteObject" message with an object identifier as an argument to CP browsers to delete the object.

3.5.2 Application level protocol extensibility

Application developers should be able to define their own message types and format, and message handlers both in the server and client side (Section 3.4.2), consistently with existing VSCP. Our approach is to leave room for application developers to extend VSCP. An application program in 3D MUSVEs is actually a set of 3D objects and actions associated with them. To invoke the action attached to the 3D object, the protocol is required to send (1) a target object identifier, (2) a method identifier to be invoked, and (3) its arguments. We can understand that the scheme is same as the message described above.

To support extensibility, the method identifier other than those pre-allocated for 3D operations can be used for application specific messages. In this case, the role of CP bureau is to correctly route the message to the AO which interprets and responds to the message. However, the CP bureau has no understanding of the message, the actual message semantics is defined by the AO.

To ensure that CP browsers are capable of interpreting application specific messages, we need to extend the CP browser's capability to respond to messages defined as extensions to the base VSCP message set. We use local scripting mechanism for this purpose (Section 3.3.1). The meaning of a message at the CP browser is defined as the following. If a CP browser receives a message to a 3D object and the object has a method for the message written in a scripting language, then the CP browser executes the method, otherwise no action is taken.

This mechanism also guarantees dynamic extension of service in MUSVE in combination with AO mechanism (Section 3.4.2). When application developers want to introduce a new service into an existing MUSVE, they can develop an AO's client-side script and server-side script by using application specific messages without being caught by pre-defined protocols. After developing the scripts, what the developer needs to do is only releasing the client-side script and VRML file in WWW server and then executing the server-side script. Then CP system automatically downloads the client-script and VRML file into the existing MUSVE in WWW manner and application specific messages are processed by this mechanism. In parallel with this message passing, awareness of objects described in the VRML file is controlled by CP bureau.

VSAP shares many protocols with VSCP. Currently, the registration protocol about a shared application is unique in VSAP for distinguishing shared applications from users.

3.6. Scalability

In MUSVE systems, scalability of the system is very important issue, especially for realizing our goal, i.e. a virtual society. Simple implementation of MUSVE system does not allow the system to scale. We summarize our approaches to scale the CP system in this section.

- Our architecture frees CP bureau from management of scene data and interpreting application specific messages. Therefore it can reduce the processing cost of the CP bureau. As described in Section 3.3.4, CP system downloads a static scene data into CP browser as a part of VRML files and the data is duplicated/shared among all CP browsers. In each CP browser, collision detection and terrain following which need high computational cost of these calculations are done and dynamic data can be managed by local script (Section 3.3.1). Application specific message passing between the local script (client-side script) and server-side script is managed by the both scripts, not by CP bureau.
- The local scripting mechanism of CP system allows offloading some processing into CP browser. This allows application developers to develop shared applications based upon event-driven approach, rather than shared state changes, to process the event in the local script. As described in Section 3.4.4, the application developers can define the event in various granularities from low level event to high level (logical) event. For example, in case of developing a shared application "flower", application developers can define high level event such as "germination" and send the event to "seed" object. When the object receives the event, it can carry out its task to germinate the seed and come into blossom by a local script. Then the flower of the seed is in bloom in all CP browsers. This also enables such techniques as dead reckoning (Singhal and Cheriton, 1995).
- As we described in Section 3.4.2, AO model allows separation between AOs and CP bureau and enables them to run in physically different computers. We can allocate dedicated computer systems for each dedicated application, such as high performance computing, database processing, and A.I. processing. This scales the CP system in physical aspect of the computer system.

These mechanisms enable reducing the burden in CP bureau and the amount of message traffics between CP bureau (or AOs) and CP browsers. However, they are not enough to realize hundred people's access to a virtual world in the Internet.

To realize the scalability, it is necessary for us to introduce a way to limit the number of messages needed between browsers to support the shared scene.

3.6.1 Consistency

The fundamental model of MUSVEs is a shared 3D space. Such a space must be seen consistently by all users in that space. Different systems provide different levels of consistency ranging from a strict interpretation to best effort algorithm (Mosberger, 1993).

In a strict interpretation of consistency, actions that occur in the shared space must be propagated to all participants in that space, and conflicts between user actions are either avoided or resolved. Furthermore, actions in the space are represented through a causal relationship so that users can understand what happened before and what happened after a certain action. Obviously, maintaining this level of consistency in a system where there are many participants is a complicated task and one that requires significant exchange of information between the copies. The choice of algorithm is crucial to the amount of message passing needed for consistency and it affects the system performance. Any distributed consistency algorithm has two major concerns:

- Membership: The membership of the consistency group, i.e. who is taking part in the consistency algorithm, is crucial to system performance. A mechanism that reduces the number of participants in the consistency group directly reduces the number of messages that must be exchanged.
- Consistency guarantee: Once the membership has been decided upon, the next issue is what model of consistency is used by the consistency algorithms. There has been a lot of research regarding the issue of distributed consistency in more traditional data applications with a goal of reducing the cost of algorithms. In this work, we tried to relax the degree of consistency either in the temporal domain, or in the data value domain.

To attack these issues, we rely on a facet of our application domain, 3D social space, and exploit a spatial area of interest (AOI) model to reduce the participants in any consistency decision. We then use adaptive techniques and a range of consistency algorithms to reduce the message traffic. In the following section, we outline the AOI model we have adopted.

3.6.2 Spatial areas of interest

In the previous experiments with the DIVE system (Carlsson and Hagsand, 1993), we have observed that participants form sub-groups where activities occur in

clusters or peer-to-peer within the global session. This mimics the way we use the spatial model in the real world. The observation can be exploited to decrease overall message passing if one can deliver packets only to the recipients they are intended for, i.e. those within the sub-group. In this way, the amount of global traffic is limited, and the number of incoming messages to each user is reduced.

This notion of a "spatial area of interest" associated with a user or object has evolved out of work in the COMIC project (Benford et al., 1993, 1994). The spatial area, known as an aura is defined to be a sub-space which effectively bounds the presence of a user or object within a given medium and which acts as an enabler of potential interaction (Fahlen, 1992). From the user's (or object's) point of view, aura is the area around it that is deemed to be interesting, and anything outside the aura is not considered interesting. In contrast, anything within the area is candidates for influence or interaction. Figure 3.14 shows how aura works. It shows that both user A and user B are in each other's aura but user C is not in any other's aura. In this case, any updates about user A, such as position update will be sent to user B, not user C. In contrast, any updates about user C will not be sent to anyone. Thus user A and user B can see each other but they can not see user C and user C can not see them. After colliding user C's aura and user A's aura, user C can see user A.

In addition to notion of aura, Benford and others (1993, 1994) defines two notions about a spatial area of interest: focus and nimbus. They represent the degree of interest users (or objects) have in each other and further subspaces within which users (or objects) choose to direct either their presence or their attention. The focus represents the degree of interest one user brings to bear on another. The nimbus represents the degree of attention one user pays to another. The combination of the focus and nimbus of two interacting users defines their level or degree of interaction.

It is the model that we seek to use to drive our consistency mechanism and to reduce the number of users in any consistency algorithm. We use this model to decide which CP browsers need to be send information and reduce the amount of message traffics and the number of users (or objects) based upon the architecture of CP system.

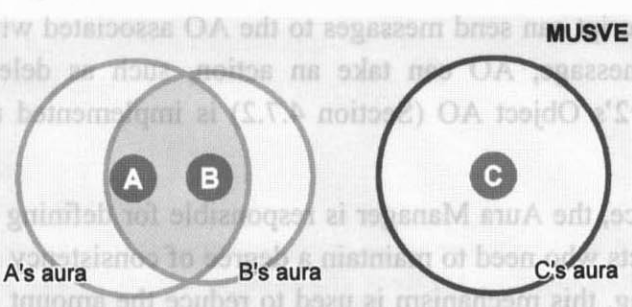

Figure 3.14 Aura mechanism

3.6.3 Implementation

To realize the mechanism, CP bureau supports the Aura Manager (Figure 3.5) which has responsibility for tracking each user's (or AO's) position and determining which aura is collided. After colliding, the Aura Manager adds both objects to a single consistency group for maintaining their shared information. The group could contain multiple users or objects simultaneously.

The amount of processing and messages generated as a result of any one CP browser moving depends on the configuration of CP bureau with respect its aura. The aura algorithm use two crucial parameters, size of aura (AURASIZE) and maximum number of avatars in the aura (MAXINAURA). The first parameter dictates the size of the bounding box (sphere) for avatar aura collision calculations at the CP bureau. The second dictates how many of the potential collisions are reported to the CP browser. If the aura size is large, but the maximum number of avatars in an aura is restricted to a small number, then a CP browser will only ever be informed of a small subset of potential collisions, where the subset equals the maximum number of CP browsers allowed in an aura. Messages sent by a CP browser are replicated to all browsers who have that browser in their aura. Hence, the MAXINAURA is a rough indication of the replication factor for each message. This mechanism also applies to AOs to scale the overall system. However, since it is implemented in CP bureau and transparent to applications, application developers do not need to be aware of the underlying mechanism when they develop the applications.

In addition, we introduced an object associated with aura with size 0 which acts as a kind of "one-way" object. Users can see the object, but the object can not see them. For example, "flower" can be implemented as a "one-way" object. It is because only its position and appearance should be shared if it does not need to accept user interaction. CP bureau can utilize the information to reduce the cost of aura calculation.

The "one-way" object can easily extend to "two-way" object by using a local script. As we described in Section 3.4.2, AO enables adding a local script to the object and the script can send messages to the AO associated with the object. After receiving the message, AO can take an action, such as delete the object. For example, PAW^2's Object AO (Section 4.7.2) is implemented using an aura with size 0.

In essence, the Aura Manager is responsible for defining groups of spatially co-located objects who need to maintain a degree of consistency. As it decreases the degree of sharing, this mechanism is used to reduce the amount of information that has to be sent out from the server as a result of any state changes.

3.7. Performance Evaluation

We evaluated the performance of CP system with a dedicated application which can simulate a user's behaviors in a MUSVE. We call it "dummy user" application. With the application, we can easily simulate a MUSVE populated by many users. The application is developed based upon our user behavior model, i.e. a user is walking around in the world with chatting and sometimes carrying out shared applications in the environment.

Therefore, the "dummy user" application provides several configuration parameters for performance evaluation as follows:

- **The number of users:** The application can be configured with any number of users: each user will generate a separate network connection to the CP bureau.
- **Movement:** For all users, the percentage of users that move per second can also be specified. This simulates user's walking. 100% and 1m/s are specified.
- **Chat messages:** For each user, the size and frequency per second of chat messages can be specified. 40 bytes per 10 seconds is specified.
- **Application specific message:** For each user, the size and number of application specific messages per second can be specified. Application specific message can simulate the user's behavior about carrying out a shared application. 256 bytes per 10 seconds are specified.
- **Attributes:** For each user, the size of the user specific attributes, managed by CP bureau, can be specified. 256 bytes (for a URL of the user's avatar and her handle name) are specified[1].

After the dummy user connects to CP bureau, she exchanges some set up messages for the network session and then stores some attribute information, such as user name and URL of avatar at the CP bureau. Then each dummy user begins to move, send chat message to simulate chatting, and send application specific messages to simulate carrying out shared application according to the "dummy user" application's configuration parameters.

[1] The attributes mechanism is introduced to reduce unnecessary network traffic between CP bureau and browsers. The CP bureau automatically sends the attributes specified by a user or application to other CP browsers which enter her/its aura without communicating with her browser or the application. In naive implementation, the CP bureau needs to communicate with her browser or the application to know, for example, the shape of her avatar, i.e. its URL and notify it to the browser which entered her aura.

3.7.1 CommunityPlace bureau

In the first set of experiments, we evaluate performance of CP bureau. We load the CP bureau with increasing the number of users. Its performance is measured for three types of computers: a Sun Sparc UltraServer 170 (Solaris 2.5 with 320MB of memory), an SGI Indigo2 (Irix 5.3, with 256MB of memory), and a Sony NEWS 5000 machine (NEWS OS 6.1 and 256MB of memory).

The "dummy user" application is running in Sony NEWS workstations connected to these computers, where the CP bureau is running, by using a 10 Mbits Ethernet connection. The Ethernet is at all times lightly loaded with general day to day network traffic. Figure 3.15 shows the CPU load.

As can be seen from Figure 3.15, on a Sony NEWS workstation, assuming a reserve of 30% CPU idle time, then the CP bureau can support 400 connections. The SGI machines, at the same load, will handle approximately 425 connections and the Sun 170, approximately 520 connections. The figures correspond roughly to the relative performance of the three machines, the Sony and SGI are comparable, and the Sun 10% more powerful. This result implies that a larger server machine would support a correspondingly larger number of connections. However, the network traffic generated by the CP bureau is also an important consideration. Figure 3.16 shows this information.

The network traffic indicates the amount of data arriving at the CP bureau from dummy users and the amount of data leaving the CP bureau heading for dummy users. Generally the data leaving the CP bureau will be greater than that which arrives, because the CP bureau is replicating the data and distributing it to CP browsers. As can be seen, the Sony NEWS will support a maximum of 500 connections, before network performance tails off, the SGI's performance is similar.

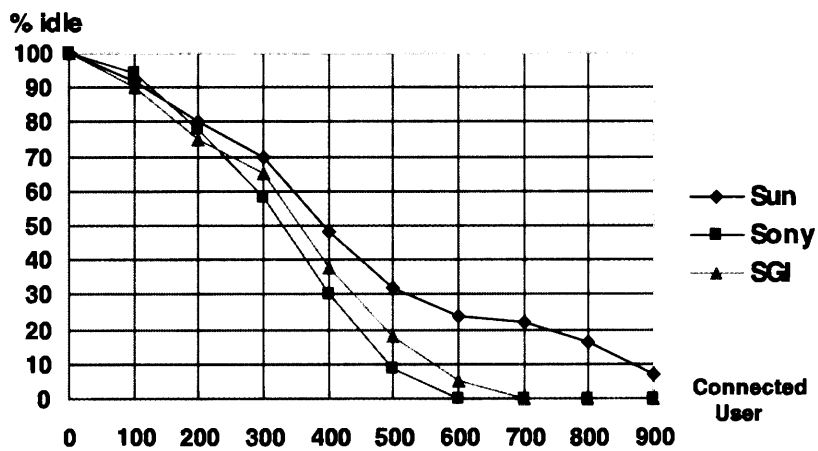

Figure 3.15 Sever CPU load

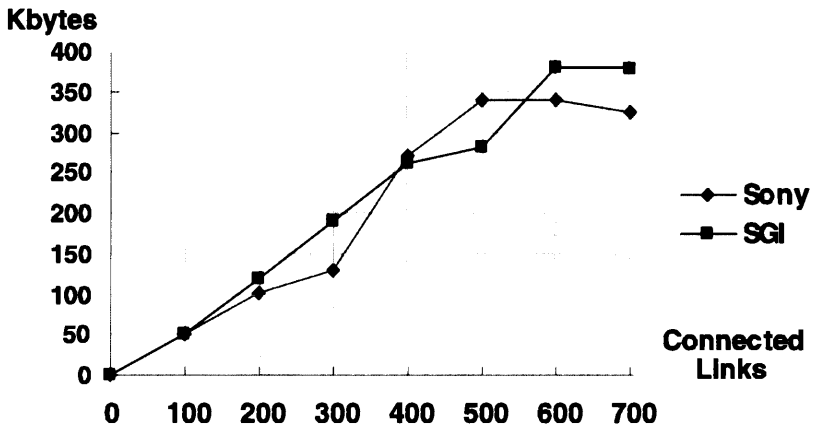

Figure 3.16 Network traffic at the server

Although not shown, the performance of the Sun machine is correspondingly better and supports approximately 630 connections. At these points, all bureaus are handling approximately 350 Kbytes of network traffic. The reader should note that for the purposes of these experiments, the configuration parameters are set to cause an artificially high amount of message traffic. Using more realistic figures collected from actual usage at our public servers, a Sun workstation will support up to 1000 connections.

3.7.2 Application model

As discussed in Section 3.6.3, the AOI algorithm uses two parameters to reduce network traffic; AURASIZE and MAXINAURA. To understand the effect of varying these parameters we present two further experiments. Figure 3.17 shows the results.

In the first, the AURASIZE is set to a higher value of 100 meters and the MAXINAURA value gradually increased. The network traffic is measured and

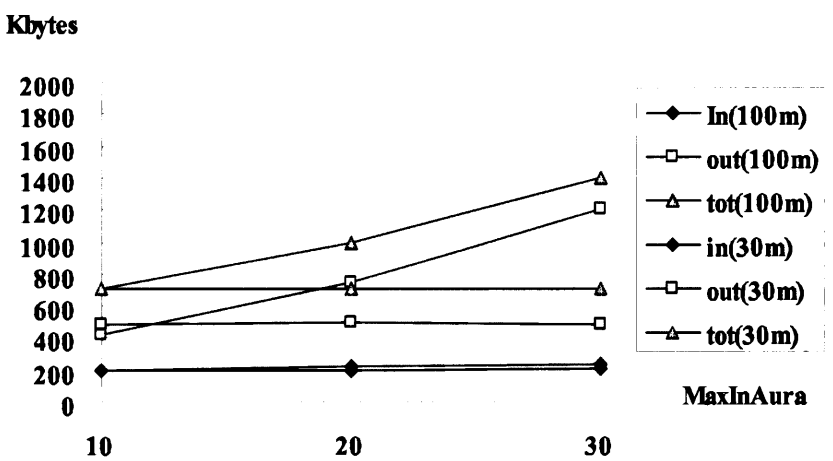

Figure 3.17 Network traffic for Aura size and MaxInAura

presented. In the second experiment, we keep the AURASIZE at 30 meters, and alter the MAXINAURA. In both cases the number of connected browsers is held constant at 100.

Looking at Figure 3.17 the total message traffic using an AURASIZE of 30 meters stays constant as the possible number of avatars in an aura increases.

In contrast, if the AURASIZE is increased to 100 meters, the effect of allowing more avatars in an aura is a marked increase in total message traffic. Looking in more detail, although the traffic from CP browsers into the CP bureau remains constant (at about 200k) the traffic generated by the CP bureau starts at 420k when MAXINAURA is 10, and climbs to 1400k when MAXINAURA is 30.

These experiments show clearly the difficulty in predicting message traffic (and hence server load) based upon the AOI algorithms. While it is clear that increasing the MAXINAURA will lead to a growth in traffic, the interplay with the AURASIZE is less clear. The actual result of these two factors is dependent on the movement behavior of the users, i.e. browsers which in turn is based upon the scene design.

3.8. Summary of this Chapter

In this chapter, we described basic system architecture and components of CP system and how it works to realize 3D MUSVE over the Internet. Its basic architecture is technically based upon not only client-server model but also peer-to-peer model to make the environment large scale to enable populating a large number of users. The CP system adopts a natural extension of the WWW system and can seamlessly fit the environment into the WWW architecture. It consists of the following three software components: CP browser, CP bureau, and AO.

The CP browser is a VRML-based multi-user browser, which provides a user interface to access MUSVE and exchanges shared information with the CP bureau and other browsers. It can visualize the 3D virtual world described in VRML, carry out local scripts with Java system, and allows a user to navigate the world and communicate with other users using on-line text chat, voice chat, and gesture-based communication.

The CP bureau is a server system for managing a MUSVE and acts as a message replicator among CP browsers. Since it is responsible for maintaining only the location and some attribute data belonging to the CP browsers (e.g. their avatar data), the total state of the shared virtual scene is split into two locations: VRML data managed at the CP browsers and limited data managed by the CP bureau. The CP bureau does not maintain a database of the virtual world. Rather, the data is

replicated at each CP browser. Also detail calculation of collision detection of avatar is done in each CP browser and the CP bureau focuses on aura calculation for consistency management. By pushing data out to the CP browsers, we are able to both reduce bureau load and increase performance, because the browser has local copies of data and is therefore not forced to access the bureau when it needs new data.

Also we described two multi-user application programming models in the CP system: SSS and AO model. These models enable providing a shared application whose appearance and behaviors are shared among users in the system. The SSS mechanism supports both fully replicated scripts and master slave scripts and allows application developers to decide which to use. The AO model implements a master model with a single copy of the data. However, application developers are free to cache data at the CP browsers and implement their own consistency algorithms.

With the dummy user application, we showed one thousand users can access the CP system simultaneously. In addition, we evaluated the network traffic at the CP bureau and network traffic for user's aura size and the maximum number of users in the aura.