Moving from MOOs to Multi-User Applications

Steven M. Drucker
Virtual Worlds Group
Microsoft Research
Microsoft Corporation

Abstract

This paper provides a brief description of the work we have done on the *V-Worlds* project, a system that facilitates the creation of multi-user applications and environments. We have taken concepts originally found in object oriented Multi-User Dungeons (MOOs) and extended them to deal with more general multi-user and in particular multi-media applications. We present reasons behind the architectural decisions of the platform and show that it has been used successfully for a wide range of examples.

Keywords

MOOs, Virtual Environments, Rapid Prototyping, Shared environments, Multiuser Applications, DIS, CSCW, Distance Learning, DHTML, ActiveX.

1. Introduction

Computers are increasingly being used as communication devices as opposed to only information processing and retrieval devices. In addition to text or voice over the Internet, entirely new modes of use are developing that allow for communication in conjunction with shared data manipulation.

Software systems are being constructed that enable synchronized modification of data at many different locations at once with the changes presented to multiple users in real time. These types of systems span a wide range of applications: simple chat systems, buddy lists, textual MUDs, graphical MUDs, distance learning applications, cooperative work systems, and more. However, the development of such applications is often hindered by the difficulty in implementing precisely the features that make them attractive - synchronous updating across multiple clients, and persistence of information across sessions.

The Virtual Worlds Platform grew out of concepts used in object-oriented multi-user dungeons (MUDs & MOOs) but have been applied to a wide variety of applications. MOOs are early examples of textual multi-user environments with persistence and the addition of convenient end-user extensibility. Many of the original decisions in the design of MOOs provide a strong basis for rapid prototyping of multi-user applications, albeit with text-only interfaces. This model faces serious limitations when the user interface is extended to include multi-media.

In this paper I will address the limitations of prior work, highlight the unique features of our current work and discuss applications of this system.

2. Background and related work

An object-oriented MUD, such as White and Curtis' MOO [1,2], is a network database server that stores objects having properties and methods. The topology of the space is defined by "room" objects, representing discrete locations, interconnected by portal objects.

Objects in a MOO can also represent things located in a room, and objects called "players" or "avatars" represent the user's character in the world and are intimately associated with a user's connection into the world. Users in the same room are able to talk by typing text and reading the text that others type. Each MUD room is superficially similar to an Internet chat room or IRC channel, but there is no persistence of objects or their behaviors within IRC. The room metaphor generalizes quite well to a variety of applications. A room could be a classroom for a distance learning application, or a team-room for cooperative work.

Other related work includes some systems exclusively focused on graphical multi-user environments including the Spline system from MERL [3], which uses a similar update region to confine the amount of information that needs to be distributed. Zyda et al [4] also discuss building multi-user graphical environments. Yet another related work is discussed in Singh et al [5]. This work does not look at rapid development or end user extensibility and is focused primarily on graphical environments.

There are now massive multiplayer gaming worlds such as Ultima Online [6] and Everquest [7] that are based on MUDs, but don't allow for end user modification or programming. Curtis et al [8] have since developed LambdaMOO themselves towards more multimedia architectures making many similar design tradeoffs that we do. Greenberg et al. provide a good summary of toolkits for synchronous collaboration for cooperative work [9].

3. Development on the V-Worlds platform

Our most fundamental departure from MUDs is the support for generalized interfaces at end-user clients. This support necessitates distribution of information updates to the extent that there is sufficient data to render graphics, sound and other UI at the local clients. Our architecture is based on extending the Microsoft Component Object Model (COM) and OLE Automation, which allows us to easily extend the system in any COM compatible language including C++ and Java, or any ActiveScripting language (VBScript, JScript, PerlScript, etc).

An earlier architecture document is discussed in Vellon et al [10]. The exemplar inheritance mechanism widely used in MOOs and also in our system is similar to that used in the SELF system [11].

Programming behavior in V-Worlds can be accomplished in three ways by: (1) defining methods attached to objects within a shared environment; (2) adding generalized DHTML to handle user interface changes on a particular client, or (3) the addition of ActiveX controls (either preexisting or special purpose) on clients that can communicate within a shared environment.

The V-Worlds Platform supports rapid prototyping in the following fundamental ways:

- UI development via DHTML/scripting
- Delegation-based dynamic object model changeable at run-time
- Automatic distribution of appropriate information and procedure calls.
- Automatic persistence of information

These are explored in further detail in the sections that follow.

3.1 DHTML/Scripting UI

The V-Worlds System Object (described in the next section) is embedded in a web page, which allows authors to rapidly develop different UIs using existing DHTML development techniques.

Dynamic HTML (DHTML) is composed of the Internet standards Hypertext Markup Language (HTML) 4.0, Cascading Style Sheets (CSS) 1.0, and the associated document object model. Taken together, these provide a powerful system for quickly building and modifying an application user interface.

By adding the V-Worlds System Object to the web page, it can interact with the rest of the web page and thereby any object accessible from within the page's document object model. DHTML can be used so that objects can be manipulated from or cause changes to the shared world. These changes can then be automatically reflected on the clients of other users as explained below.

Additional ActiveX objects, either already existing objects or specifically designed ones, can easily be added to the web page and hence be controlled through the shared world.

We have also used other containers for the V-Worlds System Object such as Visual Basic and MFC, which allowed us to exploit these rapid UI development platforms in conjunction with the v-worlds for development of multi-user applications.

3.2 Delegation-based Dynamic Object Model

To facilitate the development of new types of objects, V-Worlds implements object *inheritance* in a delegation-based fashion. A V-Worlds object has a property that references its *exemplar*. The object's exemplar is similar (but not identical) to the *class* of a C++ or Smalltalk object, and quite a bit like the SELF system [11]. When V-Worlds accesses a property or a method of an object, it first looks in the object itself for that property or method. If it does not find it there, it then looks in the object's exemplar. The search continues up the exemplar hierarchy until the property or method is found or the top of the hierarchy is reached (in which case an error results). This mechanism differs from C++'s in several ways:

- The search is done by *name*, at run-time (i.e., *late-bound*)
- An object instance can have methods and properties attached to it (beyond those introduced by its exemplar)
- An object's exemplar is, itself, another object instance
- An object's exemplar can be changed at run-time
- V-Worlds does not support multiple inheritance

Inheritance facilitates development because it allows content authors to create new objects by specializing existing ones. Having created a new object, an author can allow others to further specialize by declaring his object an exemplar and allowing others to instantiate it or create additional exemplars that inherit from it.

V-Worlds does not support multiple inheritance, primarily, to keep the programming model simple. Supporting multiple inheritance requires that users be prepared to handle unintentional name collisions and

classes encountered multiple times through different base classes (the C++ "virtual base class" problem).

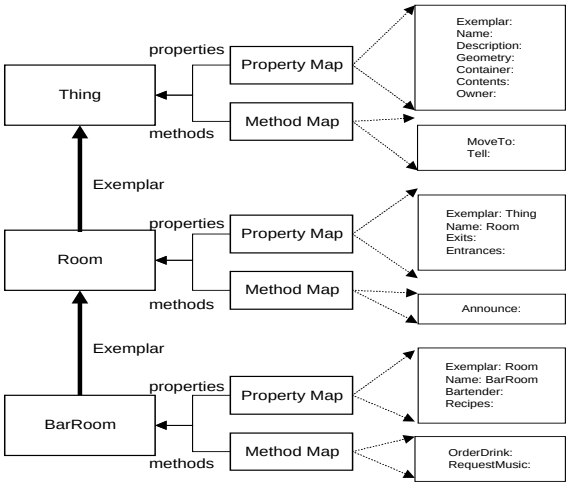

Figure 1. Property and Method Inheritance

Figure 1 illustrates the delegation-based mechanism for dispatching a reference to an object's properties and methods. It is a straightforward implementation of dynamic inheritance.

3.3 Basic Object Model

Similar to MOOs, a few basic objects are provided, such as "Rooms, Avatars," and several others. Not all of these are used in every application built on the platform. These objects, in fact, are all based on the single generic object "Thing". Users of the system can add properties and methods to instances of the objects, or change the inheritance chain dynamically.

The Thing exemplar is the parent of all objects and defines properties and methods shared by all. These include a unique ID for the object, properties that can be overridden such as the name, a reference to the exemplar parent object, and other generic information. It also defines a container object and a contents list of objects, defining a containment relationship that is used for a variety of purposes - the contents of an avatar can be its inventory of carried objects, the avatars in a room are contained in the room's contents list. This is an example of logical structure that makes the world more accessible to scripting. Thing also defines methods like MoveInto, which changes the container the object is located in.

Avatar has a variety of properties and methods to specify the object representing the user in the world. These include properties such as gender (of the avatar, not necessarily of the user), list of friends, list of users being muted, its home room, optional user information, log-in password, etc. Avatar methods include a Tell function that allows strings of text to be transmitted to the user's client, and an IsConnected property, allowing scripts to determine if the avatar is actively attached to a logged-in user. There is a one-to-one correspondence between a connected avatar and a client application.

Rooms define the topology of the world in much the same way as rooms in MUDs do. This topology is especially important to help manage the amount of information that needs to be updated for any specific change in the world. Only the avatars (and hence the clients connected with those avatars) within a certain room need to be updated synchronously with the changes that take place in that room.

3.4 Event Mechanism

In addition to providing inheritance, V-Worlds also provides an *event* mechanism that facilitates writing methods that respond to actions in the environment. V-Worlds objects support a method called *FireEvent*. This method is passed an event *name* and results in a prescribed sequence of method invocations. When `Bob.FireEvent("Foo")` is called, the following methods are invoked:

- Each of the objects in Bob's contents has its OnContainerFoo method called
- Bob's container object has its OnContentFoo method called
- Bob's OnFoo method is called
- Each of the other objects in Bob's container has its OnPeerFoo method called

Events are fired for all key V-Worlds activities: connecting and disconnecting, talking, moving, entering and exiting Rooms, user interface events, etc. This event routing mechanism allows objects to sense key activities in the environment and to respond to them.

By making this event mechanism extremely late-bound, new objects and behaviors can be introduced into the system without modifying the system itself. For instance, an object that listens to what a user says and translates it into another language has been added to the system simply by adding an object to the contents of an avatar and implementing the OnContainerTell method.

3.5 Run-time Editing

Another aspect of MUDs that we have adopted in V-Worlds is the ability to perform *live* editing of content. V-Worlds allows objects to be created and modified while those objects (and others in the same environment) are in use (on the server and connected clients).

Most Web content cannot be edited in such a manner. Web pages, for example, are usually authored off-line and then posted on public servers during times when users are not likely to be accessing them (to avoid missing pages or incorrect links during the posting process).

The live-editing capability of V-Worlds includes more than just the ability to create object instances and to modify their properties. V-Worlds allows methods and properties to be added and deleted from objects and object exemplars to be changed. As with property changes and method invocations, V-Worlds will propagate these changes to all the clients affected by the changes. (Note: in practice, these types of changes are usually made to exemplar objects and all client machines typically cache exemplar objects. Thus, changes to object structure are usually replicated in all connected clients.) In addition to replicating these changes, V-Worlds will also persist them by writing out the necessary log records.

The replication and logging of these changes occurs automatically as an object's structure is maintained in its properties. An object's exemplar is referenced by a property. An object's methods are kept in a single "map-" (dictionary-) valued property. An object's properties are kept in a single map-valued property. Thus, changes to an object's structure are really modifications to an object's properties. As V-Worlds automatically replicates and persists any property changes, this mechanism also replicates and persists changes in object structure.

The ability to change an object's structure at run-time is very valuable. First, it allows changes to be made to an environment without having to shut down access to it. Second, it allows a system to be extended by content providers and, ultimately, end-users without having to teach them about interface description language (IDL) files and recompiling a complicated system. Together, these features facilitate long-term operation, maintenance and enhancement of multi-user applications by content developers requiring less knowledge of the underlying mechanisms.

3.6 Automatic Distribution

V-Worlds is a multi-user multimedia system. Users can "enter" a world and interact with other users in the world. To facilitate the coordination of activity and the implementation of persistent world state, we chose a client/server architecture for V-Worlds. In addition to using the V-Worlds server to continuously update the clients, large "out-of-band" content (geometry, sound, other DHTML pages) is served from a standard web-server.

V-Worlds support for client-server programming is inherently built into its object model:

- Client-side V-Worlds objects "know" that they are proxies of server objects
- Client-side changes to object properties are automatically propagated to the server and to other clients (except as noted below)
- Server-side changes to object properties are automatically propagated to clients
- V-Worlds object methods can be marked as "client-side" or "server-side"(described below)
- Client-side invocations of server-side methods are automatically remoted to the server
- Server-side invocations of client-side methods are automatically remoted to clients

From the V-Worlds user's perspective ("user" here referring to a content developer using the V-Worlds SDK) the client server communication is invisible. Once the client has been connected to the server, modifications to properties are automatically replicated (to the server and other clients) and methods automatically run on the designated machine. The only awareness that is required of the user is that remoted methods are executed asynchronously (there is a way to perform synchronous client-to-server communications, but it requires explicit coding).

Because client server communications are handled automatically, it is important that unintended and unnecessary communications be avoided. V-Worlds provides a mechanism for this purpose. Properties can be marked as *local*, indicating that changes to them should not be automatically propagated.

The most important mechanism that V-Worlds provides for limiting communication needs is its *bystander* algorithm. This algorithm determines what information needs to be provided to clients and updates only this information when necessary. The bystander algorithm relies on a hierarchy of *containment* of V-Worlds objects. In the most common case, changes to an object need only be communicated with avatars (and hence the clients) in the same room as that object.

3.7 Automatic Persistence

The ability for users to exit and re-enter the environment with changes kept from one session to the next is of fundamental importance in multi-user communication systems. In addition, modifications

made by other users while a particular user is not presence are also important.

V-Worlds implements persistence by allowing entire objects to be marshaled into a serial stream and by automatically *logging* changes to object properties.

Storing the state of an entire object is relatively straightforward - V-Worlds stores the values of its properties and a record of what methods it has.

V-Worlds automatically logs changes to object properties. When a property value is changed on the server, the server automatically records the change in a log file. This file is a simple sequential file. To restore the state of an environment, V-Worlds reads this log, reapplying the property changes. If the server crashes, only the unwritten change records are lost (although expensive, the server can be told to immediately write changes out to the log file in order to provide the maximum robustness).

To avoid unnecessary logging, V-Worlds allows properties to be marked as *volatile* indicating that changes to them not be logged.

To avoid large log files, V-Worlds can write out its entire state to a new log file (by writing out complete objects) and then the old file can be deleted (or archived).

4. Implementation Details

Providing a comprehensive description of how V-Worlds works would exceed the objectives of this paper. There are a few implementation aspects, however, that are worth noting, specifically a description of the IThing interface, the object that enables the dynamic object model; its relationship with scripting; and the embedding of the V-Worlds object within a web page.

4.1 *IThing* Interface

As mentioned earlier in this paper, V-Worlds is implemented on top of COM. At the heart of V-Worlds is the *IThing* interface. All V-Worlds objects (Avatars, Rooms, Artifacts, etc.), from the COM perspective, are instances of *IThing*. The *IThing* interface provides much of the key functionality of V-Worlds:

- The ability to add and delete methods and properties to an object at run-time
- The ability to access methods and properties, taking object inheritance into account
- Object-level persistence (serializing a whole object)
- The low-level properties required of all objects (*exemplar*, *owner*, etc.)
- Easy access via OLE Automation

From a C++ perspective, *IThing* is straightforward. Methods and properties are added by calling *AddProperty* and *AddMethod.* Properties are read and written to by calling *get_Property* and *put_Property*. Methods are invoked by calling *InvokeMethod*. Note that access to properties and methods is through helper functions. These helper functions are key to providing inheritance and the ability to dynamically modify objects at run-time. Rather than binding statically (during compilation), accessing properties and methods through these helper functions allows V-Worlds to perform late binding. The helper functions also enforce V-Worlds security policies and automatically perform any remoting (e.g. replicating property changes or invoking remote methods). On the server, the *put_Property* helper function is responsible for logging property changes.

4.2 Scripting IThing

From scripting languages (and anything else that uses OLE Automation), access to V-Worlds objects is even easier. V-Worlds *IThing* objects implement *IDispatch* by consulting the dynamically added properties and methods in addition to the static OLE TypeLib information. Essentially, the implementation of IDispatch turns an *x.y* reference into an `x.get_Property("y"),x.put_Property("y")` or `x.InvokeMethod("y")` helper function call. Thus, the content developer can more naturally access added methods and properties:

```
' In VBScript

' add a new property and initialize it
foo.AddProperty "Age", 12

' access the property
DogYears = foo.Age * 7
foo.Age = DogYears

' add a new method
bServerSide = True
set method = world.CreateMethod(…,
    bServerSide, …)
foo.AddMethod "newmethod", method

' call it
foo.newmethod 7, "Bob"
```

4.3 Embedding V-Worlds in a web page

As mentioned previously, development using the V-Worlds Platform exploits the ability of objects to be embedded in web pages. Actually, the V-Worlds system object can be embedded in any ActiveX container

application. By embedding it within a web page, we get the benefits of DHTML and the web Document Object Model. Once the V-Worlds System Object is in the web page, users connect using it to the shared world. Once connected, this object can communicate with the rest of the object model on the web page.

Here is some HTML code for a typical V-Worlds web page:

```
<html>
<head>

<object id="theclient"
      classid="clsid:…">
</object>

<SCRiPT LANGUAGE="VBScript">

sub window_onload
      logon "steven", password, worldURL
end sub

function logon(name, password, vws_url)
      dim world, user

      set world =
window.theclient.VWClient.Connect(vws_url)
      set user = world.Connect(name,
password)

      ' handle error codes here ....
end function

'Handle UI events generated from the client
sub theclient_OnUIEvent(who, what, args)

      if what = "OnTell" then
              HandleTell args.property(0),
args.property(1), args.property(2),
args.property(3)
      elseif
      if what = "Help" then
              HandleHelp who, what, args
      elseif what = "LookAt" then
              OnLookAt who
      elseif what = "ShowHTML" then
      elseif what = "ShowURL" then
      else

      End If

end sub

sub theclient_OnDisconnect(theworld)
end sub
.
.
.

</SCRIPT>

</head>
<BODY>
<IFRAME BORDER=0 FRAMESPACING=0
MARGINHEIGHT=0 MARGINWIDTH=0 name="mainwin"
scrolling="no" noresize height="100%" width
="100%" frameborder="no" style="DISPLAY:
none">
</IFRAME>
</BODY>
</html>
```

5. Sample Applications

In this section, we discuss five examples that have all been written on top of the Virtual Worlds Platform that demonstrate the wide variety of applications and interfaces that are possible using the underlying platform. The applications are:

- Generic Graphical V-Worlds SDK – a publicly available research platform
- The Virtual Conference Room Project – a research system for group communication
- The "Hutch" – a social support system
- "Flatland" – a distance learning system
- vMSR – a buddylist/application whiteboard system

5.1 Generic Graphical V-Worlds SDK

The generic V-Worlds SDK as it is available publicly over the web (http://vworlds.research.microsoft.com/) is currently geared toward the creation of shared graphical environments. The Graphics Viewer ActiveX control uses Direct3D RM to exploit hardware acceleration and is embedded in the web page along with the V-Worlds System Object. Events for other user's entry and exit from the current room are captured by the V-Worlds System Object and forwarded (along with pertinent information about the objects geometry and position) to the Graphics Viewer Control. The Graphics Control then loads geometry via a standard http server and places geometry within the scene. The control also intercepts mouse and keyboard events, which update the current users position and orientation. These properties are automatically sent to other users in the same room as the user and their displays are updated as appropriate.

The Graphics Control also loads a 2.5D boundary representation to help manage collision detection as the avatars move around the space. Currently, avatars are represented as 2D sprites primarily because of the ease in which end-users can customize the appearance of their avatar.

The SDK is shipped with a number of wizards that facilitate the creation of rooms and objects along with sample content and simple editors for positioning objects. In addition, an object browser allows the on-the-fly addition and modification of properties and methods.

5.2 The Virtual Conference Room Project

The Virtual Conference Room Project is an experimental research project where we are examining a better representation for characters to communicate within a graphical environment. In this case, more sophisticated behaviors have been programmed in COM and attached to the avatars within the shared world. In addition, a camera system watches the user and tries to use a combination between a user's own gestures and high-level behaviors to control a user's avatar. The camera system was developed completely independently and was added into the system in only a few hours.

5.3 HUTCH

The "Hutch" project is a collaboration with the Fred Hutchinson Cancer Research Center in Seattle, Washington. The project explores the role that shared environments can play in social support systems. Originally, the Hutch interface was heavily focused on immersive, 3D synchronous interaction. However, it was found after performing user and site testing that a less immersive interface was desired. A subsequent iteration of the interface was designed. In this interface, a 3D overview of the hospital allows the user to see both areas of interest in the hospital space and other users accessing that info. This configuration allows for serendipitous encounters with other people in the space while attending to other activities such as email and web browsing.

5.4 Flatland

"Flatland" is a distance learning application built on top of the V-Worlds platform. Instead of a 3D graphics window as in the previous examples, streaming video presentation in the form of a NetShow control is embedded in the page along with the presenter's slides.

Different layouts can easily be created, including specialized layouts for the presenter that show additional information than a typical client. Participants can ask questions back to the presenter which can be put in a question queue for the presenter to respond to. Users can also vote on polls with immediate feedback. Users can also raise hands or chat with other people in the audience. Because of latency in the streaming video, we needed to pay special attention to synchronization of events between the streaming video and other channels of information.

The development of Flatland followed a strict model-view controller separation [12,13] to assist in programming on top of the V-Worlds platform. The model was kept entirely in-world using the V-Worlds Object Model, while the view controller was implemented entirely within DHTML.

5.5 vMSR

The vMSR application connects two shared virtual worlds together. One shared world is run purely local to a single computer and allows different applications to easily share information with each other. Applications can in turn communicate information amongst many users via the global shared world. We easily prototyped a buddy list application that accesses information about a user's activities and selectively shares that with other users. Using this kind of information, one user can request that a message be sent to another individual as soon as the other user is not actively engaged in another activity. We were also interested in experimenting in a variety of different interfaces for buddy lists and chat systems and a number of different prototypes have been developed using this framework.

6. Summary

The Virtual Worlds Group has implemented a platform that facilitates the development of distributed multi-user applications. We started from the standpoint that MOOs provide a good framework for rapid development of applications. In extending that idea to multimedia architectures, we incorporated concepts from DIS and other graphical based VR systems, and from current architectures for UI development. We found the following aspects of our system design to be the most valuable for rapid development of multi-user applications:

- Dynamic delegation-based object model: This allowed new authors to significantly leverage objects built for other applications. Being able to add properties on the fly was particularly important for interactive debugging of the interfaces.
- Automatic distribution and persistence of information: By freeing the author from having to worry about the particulars of what was distributed and when, the authors could concentrate more on the interface behavior.
- Leveraging COM, OLE Automation, and DHTML: This made it possible to incorporate existing applications or easily implement new interfaces in a variety of different languages.

Version 1.1 of the platform (oriented to graphical environments and described in Section 4.1) is available at (http://vworlds.research.microsoft.com/). Later versions are being planned that better optimize server performance and that better document the API for custom UI development.

7. Acknowledgements

The Virtual Worlds Platform and examples described in this paper represent the efforts of a large group of people over several years. The author wishes to acknowledge the efforts of the entire V-Worlds team, past and present and several other teams on the various projects. In particular, Don Mitchell, Manny Vellon and Kirk Marple from the Virtual Worlds Group, the following members of the Graphics Group for the Conference Room Project (Michael Cohen, ZiCheng Liu) along with Alex Colburn, Jim Mahoney.; members of the Collaboration & Education Group (Anoop Gupta, Steve White, Jonathan Grudin), along with Greg Kimberly and Harry Chesley for work on Flatland; Lili Cheng, Sean Kelly and Chris Liles for work on the Hutch; and members of the DTAS Group (Eric Horvitz, David Hovel) along with Dave Vronay, Harry Chesley and Lili Cheng for work on vMSR. Extra thanks to Alex Colburn for the DHTML example in section 3.3. and for comments from Marc Smith, Matt Conway, Jim Mahoney, Harry Chesley, Dave Vronay and Dianne Berkeley.

8. References

1. Curtis, P. Mudding: Social Phenomena in Text-Based Virtual Realities, Intertek Vol 3.3 1992.
2. lambda.parc.xerox.com:8888
3. David B. Anderson, John W. Barrus, John H. Howard, Charles Rich, Chia Shen, Richard C. Waters *Building Multi-User Interactive Multimedia Environments at MER.* IEEE MultiMedia, 2(4):77-82, Winter 1995.
4. Zyda, Michael J. D.R. Pratt, JG Monahan, K.P. Wilson, NPSNET: Constructing a 3D Virtual World. Proceedings of 1992 Symposium on Interactive 3D Graphics. Computer Graphics. 1992.
5. Curtis, Pavel, M. Dixon, The Jupiter Audio/Video Architecture: Secure Multimedia in Network Places Ron Frederick, and David Nichols. Proceedings of the 1995 ACM Multimedia Conference. 1995
6. Singh, G. L. Serra, W. Png, and H. Ng, BrickNet: A Software Toolkit for Network-Based Virtual Worlds. *Presence* Vol 3.No. 1. 1994.
7. http://www.ultimaonline.com/
8. http://www.station.sony.com/everquest/
9. Greenberg S. and Roseman M.. Groupware Toolkits for Synchronous Work. in M. Beaudouin-Lafon (ed.), Trends in CSCW, John Wiley & Sons Ltd.. 1998.
10. Vellon, M., K. Marple, D. Mitchell, and S. Drucker. The Architecture of a Distributed Virtual Worlds System. Proc. of the 4th Conference on Object-Oriented Technologies and Systems (COOTS). April, 1998.
11. Ungar D., Randall B. Smith , Self: The power of simplicity. A*CM SIGPLAN Notices*, Vol. 22, No. 12 (Dec. 1987), in: OOPSLA '87. pp. 227-242
12. Glenn E. Krasner and Steven T. Pope, A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80, Journal of Object-Oriented Programming, 1(3), Aug./Sept., 1988.
13. Isaacs, E.A., Morris, T., and Rodriquez, T.K., A Forum For Supporting Interactive Presentations to Distributed Audiences, Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW '94), October, 1994, Chapel Hill, NC.