INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

Université d'Ottawa · University of Ottawa

JETS:
DESIGN AND IMPLEMENTATION
OF A
JAVA-ENABLED TELELEARNING SYSTEM

by

Shervin Shirmohammadi, B.A.SC.

A thesis submitted to the
School of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Applied Science
in
Electrical Engineering

Ottawa-Carleton Institute for Electrical Engineering

Department of Electrical Engineering
Faculty of Engineering
University of Ottawa

Abstract

Telelearning systems, in which students and instructors share multimedia documents in real time, have been a subject of interest for many years. Recent advancements in the Internet technology such as the emergence of Java-enabled browsers, VRML, Active X, and similar technologies have provided users with access to dynamic multimedia content and applications on the World Wide Web. One seems to be justified to take advantage of these available and widely-used technologies for today's interactive systems such as telelearning environments. In this thesis, a platform-independent, generic, client-server model for multi-user/collaborative applications through the Internet with emphasis on telelearning is proposed. The approach used to design the system ensures the operability and compatibility of the system with existing technologies and guaranties its accessibility by the majority of Internet users.

Acknowledgments

I would like to sincerely thank my research supervisor and professor, Dr. Nicolas D. Georganas, for a number of reasons. First, for assigning to me such interesting and challenging research topic; namely telelearning. Second for giving me the opportunity to experiment with different leading edge software and hardware technologies, such as VRML, Java, SUN Ultra, and so on. Finally, for his motivation, guidance, experience, and encouragement which were the most important factors that helped me finish my M.A.SC. program in four consecutive semesters.

I would also like to acknowledge the support and sponsorship of the Telelearning Research Network of Centers of Excellence who provided the necessary funds for this project, as well as the Ontario Graduate Scholarship Program for granting me an OGS scholarship for 1996/97.

Last, but definitely not least, I want to thank my parents for their constant support and encouragement, specially my mother who talked me into studying at the graduate level. Without their careful and long term planning, this whole thing would never had happened.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AAL: ATM Adaptation Layer

ATM: Asynchronous Transfer Mode

CCD: Client-to-Client Delay

CTA: Common Training Architecture

DELTA: Developing European Learning Through Technological Advance

ECOLE: European Collaborative Open Learning Environment

DSD: Delay Sensitive Data

GUI: Graphical User Interface

HTML: Hyper Text Markup Language

I/O: Input / Output

IP: Internet Protocol

IPC: Inter-Process Communication

JDK: Java Development Kit

JETS: Java-Enabled Telelearning System

JIT: Just-In-Time (compilers)

LAN: Local Area Network

MIME: Multipurpose Internet Mail Extensions

MMCF: Multimedia Communications Forum

OCRInet: Ottawa-Carlton Research Institute network

ODP: Open Distributed Processing

OISE: Ontario Institute for Studies on Education

QoS: Quality of Service

RMI: Remote Method Invocation

TL-NCE: TeleLearning Network of Centers of Excellence

URL: Universal Resource Locator

VR: Virtual Reality

VRML: Virtual Reality Modeling Language

Conventions Used in the Thesis

`Courier New Font:`

- standard (core) Java packages and classes;
- standard Java methods; methods end with (), e.g. `run()`.

Bold:

- some titles;
- Java classes or methods of the tele-learning system, when they are first introduced.

Italic:

- new terms where they are first defined.

Chapter 1. Introduction

Tele-learning has been a subject of interest for many years. The idea of a *virtual classroom* where students and instructors share multimedia material from their computers without being actually present at the same location is a very appealing subject for research. Figure 1 depicts a simplistic representation of a virtual class-room.

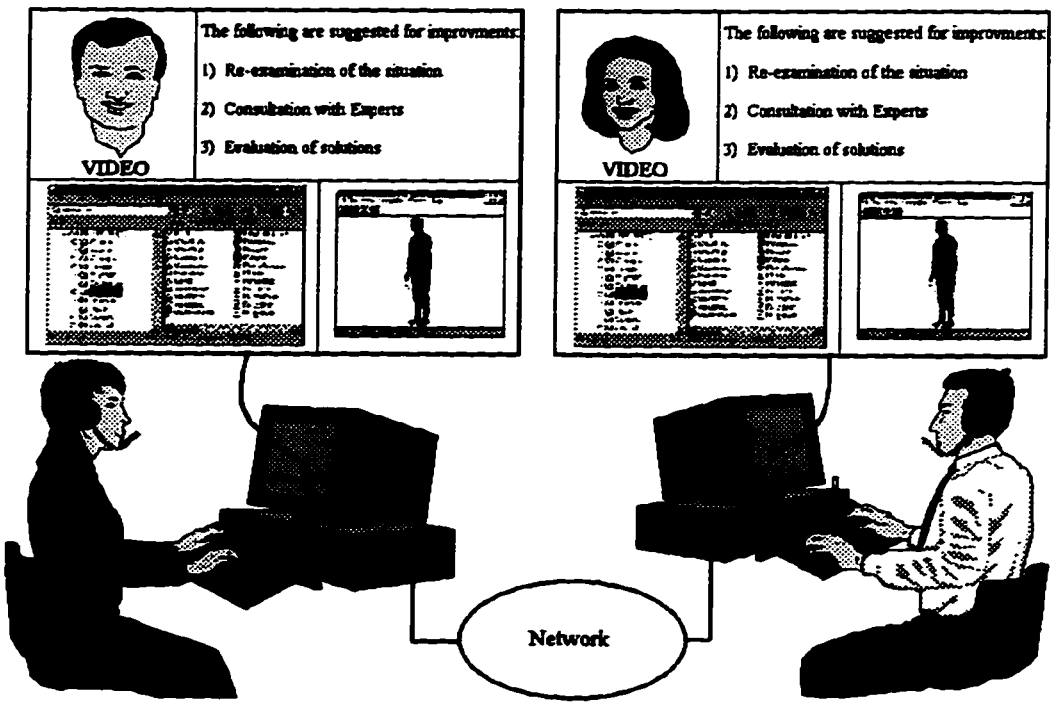


Figure 1. A virtual classroom

As one can see, the idea here is to share multimedia documents as well as to see and speak to other participants. Up until now, the progress in the development of such virtual classroom has been limited mainly by the lack of the following factors:

- high speed networks;
- powerful workstations for ordinary users;

- network access for ordinary users;
- a standard interface for presenting multimedia material;
- a standard for multiuser applications.

Some of these problems are not as significant as they used to be a few years ago. Workstations today are quite powerful and affordable for ordinary users. Also recently, companies and organizations known as Internet providers are offering affordable network access for households. As a result of both affordable powerful workstations and network access, the number of users joining the Web community is increasing at a tremendous rate. It is anticipated that Internet access will be part of the standard utilities of a household, as are electricity, water, cable television, and telephone.

These improvements in technology have stimulated significant global activity on tele-learning over high-speed networks. The European Commission, for example, has a strategic research and development program called DELTA (Developing European Learning Through Technological Advance), that involves major industrial and government organizations. Two main projects in DELTA are the CTA (Common Training Architecture) and ECOLE (European Collaborative Open Learning Environment) [26]. These projects involve both field trials, multimedia tool development, multimedia communication platforms and technical and administrative coordination.

In Canada, OCRInet, sponsored by the Ottawa Carleton Research Institute (OCRI) and industry, is the first wide-area ATM (Asynchronous Transfer Mode) broadband test bed [3]. It launched its operations in Ottawa on January 1994, with a collaborative work application over video conferencing, between BNR and the Multimedia Communications Research Laboratory (MCRLab) at the university of Ottawa. In addition to its high-speed

networking, OCRInet currently links 12 industrial, government labs, and academic sites in the Ottawa area. This makes OCRInet the perfect set-up for research about knowledge sharing, distance education, tele-learning, collaborative work, as well as a test bed for running and evaluating the performance of high-speed multimedia communications applications, such as the research project of the TeleLearning Network of Centers of Excellence (TL-NCE).

TL-NCE represents an unprecedented collection of Canadian researchers and the potential for remarkable advances in Canadian education and training. The goal is to bring computer-supported environments, artificial intelligence, high-performance networks, multimedia and collaborative tools into coherent systems. TeleLearning, which makes use of multimedia learning environments based on computers linked by the information highway, is a technology and a social innovation of fundamental importance for education and training at all levels in a knowledge-based, learning society. TeleLearning extends access and brings high quality education to all citizens, regardless of their location, age, or status.

The starting point for the TL-NCE research is a number of Canadian beacon technologies:

- TeleCSILE, a wide-area system built around a hypermedia database constructed by the Ontario Institute for Studies on Education (OISE);
- VIRTUAL-U, a networked multimedia environment specifically customized for educational activities and course delivery with field tests running at universities and workplace education sites across Canada;
- TELEFORM, a set of tools for delivery of TeleLearning in the workplace and at home.

Work at he LICEF laboratory of Quebec's Tele-université is complemented by technologies resulting from Ontario's Telepresence project and research at the

University of Ottawa MCRLab to provide coherent learning environments for workspace;

- CADRE, a set of design and authoring tools and methodology on work at the Universities of Guelf and Waterloo, the LICEF laboratory at Tele-université, intelligent tutoring work at Université de Montreal, University of Saskatchewan and Alberta Research Council.

The responsibility of the MCRLab is TL-NCE sub-project 6.1.5: "Multimedia Tele-learning over ATM Networks : architectures, application development, experiments and performance evaluation over OCRInet and CANARIE". This project is lead by Dr. Nicolas Georganas with Drs. Dan Ionescu and Emil Petriu as principal investigators. The first client group will be Professors J. Houseman and A. Morin, department of biology, University of Ottawa and their students. They have developed a multimedia course application in Zoology, which we intend to enhance with 3D objects and the ability to browse and manipulate those objects using 3D and Virtual Reality tools. Eventually, participants in that course will be able to share 3D objects and course material as in a virtual classroom.

The goal of the research for this specific thesis is the design and implementation of multimedia collaborative tele-learning applications over high-speed networks. In other words, a virtual classroom must be created where users from distant locations can access *and* collaboratively share various applications incorporating different media such as image, text, and 3D objects in real time. The emphasis in this research is on shared browsing and manipulation of 3-D objects, as well as sharing a "whiteboard", documents containing text and images, and a means of sending textual messages. The video and audio-conferencing

aspect of a virtual classroom will not be covered in this research since they are beyond the scope of this thesis. Live video and audio-conferencing is a separate research issue on its own and is being tackled by many researches and organizations in that field.

It is important to realize that because of the project's nature which deals with learning and education, it is extremely important that the infrastructure of the system be designed in a way that makes it accessible for everybody: a teacher at school with an IBM computer, students at home with PC's and Macintoshes, a supervisor at an organization with SUN Station or Silicon Graphics station, and so on. This creates a problem right from the beginning:

Since the education community, including students, teachers, instructors, professors, as well as private or funded educational organizations, is very large, the use of a specific operating system or platform should be avoided. It would be unrealistic to assume that all of these users will migrate to a specific platform just to use our tele-learning system. Therefore, it immediately becomes apparent that the tele-learning system must be platform-independent at least at the user's end.

In addition to being platform-independent, the design and implementation of the architectures and applications must be compatible and interoperable with the current existing technologies and infrastructure. Adherence to standards is very important for a tele-learning system since it will further ensure that the system can communicate to as many people as possible. The standards of interest to this research are standards for:

- networking;
- 3D object representation;

- interface for presenting multimedia material ;
- virtual classroom engine for multiuser applications.

In terms of networking, the choice seems to be clear: the Internet. As mentioned earlier, the Internet, which utilizes IP as its networking protocol, is growing at a tremendous rate. It is now possible for the average person to obtain connection to the Internet at a reasonable cost; although not at a satisfactory network speed to deliver multimedia content in real time. However, both the cost and the network speed are improving rapidly as technologies improve in that field. For example, ATM promises to deliver multimedia content at a given quality of service, including high bit-rate and low loss-rate. The high bit-rate of ATM, which can be in the range of gigabits per second, will facilitate the transfer of broadband traffic.

It is worth mentioning though, that ATM technology itself does not have a networking protocol similar to IP. Furthermore, in order to be successful, ATM must also adhere to the already established and widely used network standards; which is IP in this case. That's why in the practical implementations of ATM, the user stations still run IP as their network protocol and the IP to ATM translation is performed by ATM ARP[1]. This is also how the OCRInet is implemented [17]. In some cases, even the data-link and the physical layers are implemented by some legacy LAN like Ethernet or Token Ring. In this case the transition from LAN to ATM is performed at a special ATM switch that acts as a gateway. In any case, IP has well established itself as the default networking protocol and that's why it was chosen for this project.

[1] Address Resolution Protocol (ARP) is a protocol that translates addresses from one standard to the other. For example, when an application requests to send data to IP address 137.122.20.163, the ARP informs the underlying ATM network to use VP/VC 1/203 to deliver that data.

The other three standards mentioned above will be discussed in this thesis. Chapter 2 will briefly introduce VRML, the de-facto standard for 3D object representation on the Internet, with a short discussion on virtual reality standards; while chapter 3 will present Java, a new approach to programming for the Internet. Chapter 4 discusses the design of the tele-learning software engine including the generic client-server model. Chapter 5 presents the implementation of some sample applications including a 3D object viewer, chapter 6 contains the performance evaluation of the system, and chapter 7 concludes the thesis.

The work in this thesis contains some original contributions including:

- design and creation of a platform-independent, real-time collaborative environment;
- design and implementation of a generic client-server model for shared, real-time applications through the Internet;
- creation of generic Java classes that form the infrastructure of any collaborative environment and can be extended to implement specific client-server applications;
- implementation of a working prototype of the system.

There is one research presentation resulting from this thesis: "JETS: A Java-Enabled Telelearning System" which was presented at the 1996 TeleLearning Research Network Conference held on November 5-7, 1996 in Montreal, Quebec. In addition, the prototype of the JETS system was exhibited in a demo session at the same conference as well as in the 1996 CASCON Conference held on November 12-14, 1996 in Toronto, Ontario. A paper was also submitted at the 1997 IEEE International Conference on Multimedia Computing and Systems to be held on June 2-6 in Ottawa, Canada.

Chapter 2. VRML: Virtual Reality Standard for the Internet

It was decided to incorporate 3D objects in the context of this project since these objects are one of the most important media in tele-learning. A 3D object is not a picture or image but a three-dimensional computer generated representation of a real object that can be interacted with as in the real world. It is the medium of interest to virtual reality.

The Internet standard for 3D object representation is the Virtual Reality Modeling Language (VRML). In order to understand VRML, a concise discussion about virtual reality itself is inevitable. This is the purpose of the following section.

2.1 Virtual Reality

The basic goal of virtual reality (VR) is to produce an environment that is indistinguishable from reality in which certain things can be done or experienced. For example, a "virtual conference room" would be an environment where upon entering, one can see other participants, interact with them, and exchange information just as one would do in a real conference room.

Virtual Reality is considered to be the ultimate goal of communications: to overcome the distance barrier. When the telephone was first invented, this goal was realized for audio. Similarly, television overcame the distance barrier for video. The ultimate objective is to overcome the distance barrier not just for audio and video but for all human perceptions: vision, sound, touch, taste and smell. In other words, communications must enable users to feel as if they are physically present in a distant environment where in fact they are not. This feeling of being physically present in some location is referred to as Virtual Reality.

The idea of virtual reality may be considered to have been conceived by Van Sutherland when he introduced the idea of expressing pictures in 3D form in 1965 and proposed the use of "Head Mount Display" in 1968 [7]. A paper, published in 1972 by D.L. Vickers, one of Sutherland's colleagues, describes an interactive computer graphics system utilizing a head-mounted display and wand. The display, worn like a pair of eyeglasses, gives an illusion to the observer that he/she is surrounded by three-dimensional, computer-generated objects. The challenge of VR is to make those objects seem as real as possible in many aspects like appearance, behavior, and quality of interaction between the objects and the user/environment.

Complete immersion into a virtual reality environment requires the use of sophisticated and expensive equipment such as special goggles, headgear, and gloves (see below picture) [4]. This is one of the reasons that research and development in the field of VR has not been too impressive.

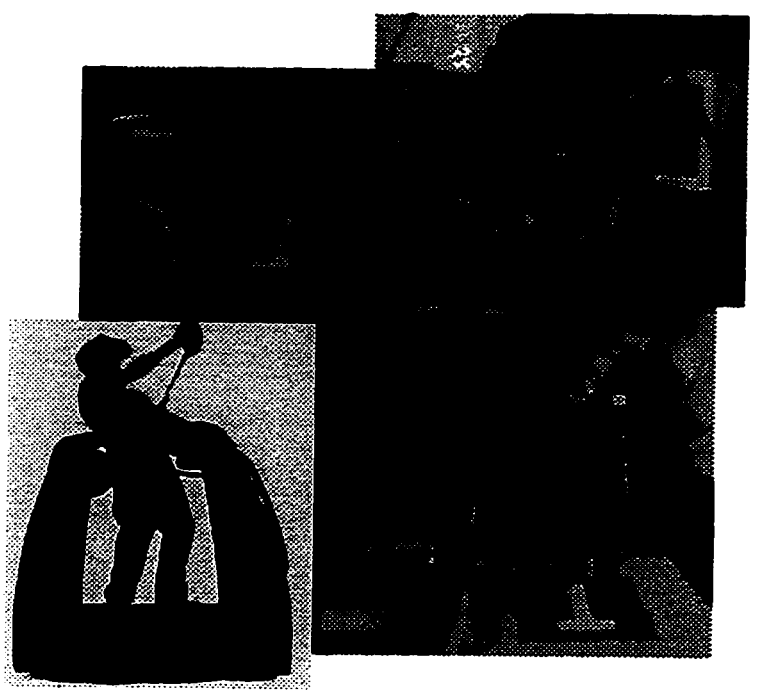


Figure 2. Virtual Reality Equipment

However, recent efforts have brought virtual reality to ordinary desktop computers, allowing users to use a typical simple mouse to interact with the 3D world on their screen. This simplistic approach has boosted up the development of various VR based applications on the Internet and has resulted in rapid development in the field of VR. Consequently, VR has started to become available to general Internet users. Just within the last year, we witnessed the emergence of simple 3D browsers leading to more sophisticated VR applications such as *QuickTime VR* and *RealVR Traveler.*

QuickTime VR is a virtual reality software based upon Apple's QuickTime Technology. QuickTime VR allows interaction with panoramas and object movies[2] [19].

RealVR Traveler, from RealSpace Inc., is a virtual reality player that combines panoramic viewing with VRML rendering, compositing and URL linking. Resembling QuickTime VR, the RealVR Traveler shows photographic panoramas with sprites and VRML objects moving around within the panoramas [20].

Both of the above applications are currently available as a plug-in for Netscape Navigator on the Power Macintosh and Windows 95/NT.

The other problem in the field of VR has been the fact that everybody was working on it independently. Since too many people independently worked on it, they all had their own ideas on how things should be done; therefore, everyone did it differently and too many standards appeared. Today, however, the VR community joined by the Internet is working together to come up with very few VR standards. Eventually, one or two main standards will emerge that will be accepted by the majority. The successful standard must cover all of the following issues in an efficient manner:

---

[2] Panoramas and object movies are scenes in which the user can look in almost any direction.

Rendering

This is the interface to the user, the graphical way in which the virtual environment[3] is presented to the user. A successful VR language must define a well-defined rendering system.

Perception Standards

The main area of concern for a VR standard is perception; after all, perception is what makes the real world so real. Therefore, a VR language has to deal with the following perceptions:

- **Touch:** This occurs when two objects "collide".
- **Sound:** When two objects are aware that they are in the same environment, they can produce sounds detectable by each other. Sound can be sent (talking) or received (hearing).
- **Vision:** An object in the same environment as other objects is able to render itself to other objects in order to produce views of itself.
- **Smell:** Currently, an appropriate hardware technology for producing smells doesn't exist nor is there an immediate need for such technology at the present; however, provisions for smell must also be made in a VR standard.
- **Taste:** Like smell, there is no immediate need for this perception; however, in the future, VR systems must carry smell and taste to fully support the idea of virtual reality.

Networking

Networking is also a very important issue in virtual reality. There must be a way for the objects in a virtual environment to utilize the underlying networking protocols to exchange information with remote objets; furthermore, the environment itself must be able to fetch remote objects.

---

[3] A virtual environment is a 3D environment that contains objects of different media types including other 3D environments or 3D objects .

Scripting Language

There is an absolute need for a scripting language in order for the standard to be generic. A Scripting language is used to describe specific properties such as:

Behavior: Components of a virtual world should be able to demonstrate behavior on their own or in response to a user interaction. For example, a fish in an aquarium must be able to swim by itself as well as swim away from a user who is knocking on the aquarium's glass.

Synchronization: In a distributed system, it is difficult to make sure that everything that is supposed to happen synchronously does so. Hence, there is a need to specify scenarios in a VR standard.

Multi-Participation: One of the most important issues of virtual reality is how to allow a virtual world to be shared by many users at the same time. Multi-user support requires support for access control and consistency.

Next, we will see how VRML addresses these issues.

2.2 VRML (Virtual Reality Modeling Language)

2.2.1 Introduction

VRML is the most widely accepted standard for virtual reality on the Internet. Mark Pesce, one of the creators of VRML, describes it as follows:

"The Virtual Reality Modeling Language is a language for describing multi-participant interactive virtual worlds connected via the global Internet. All aspects of virtual world *display*, *interaction* and *inter-networking* can be specified using VRML." [15]

Similar to the Hyper Text Markup Language (HTML), which is a language for the World Wide Web text documents, the Virtual Reality Modeling Language is a language for virtual worlds. VRML was developed as an answer to the problem of a VR interface to WWW. The following is a brief history of VRML.

2.2.2 History

VRML was conceived in the spring of 1994 at the first annual World Wide Web Conference in Geneva, Switzerland. Tim Berners-Lee, developer of WWW, and Dave Raggett organized a Birds-of-a-Feather (BOF) session to discuss Virtual Reality interfaces to the World Wide Web. Attendees agreed on the need for these tools to have a common language for specifying 3D scene description and WWW hyperlinks -- an analog of HTML for virtual reality. The term Virtual Reality Markup Language was coined, and the group resolved to begin specification work after the conference. The word 'Markup' was later changed to 'Modeling' to reflect the graphical nature of VRML. ... The group quickly agreed upon a set of requirements for the first version. After much deliberation the list came to a consensus: the Open Inventor ASCII File Format from Silicon Graphics Inc. The Inventor File Format supports complete descriptions of 3D scenes with polygonally rendered objects, lighting, materials, ambient properties and realism effects. A subset of the Inventor File Format, with extensions to support networking, forms the basis of VRML. Gavin Bell of Silicon Graphics has adapted the Inventor File Format for VRML, with design input from the mailing list. SGI has publicly stated that the file format is available for use in the open market, and have contributed a file format parser into the public domain to bootstrap VRML viewer development. [21]

2.2.3 VRML Specifications: Defining Virtual Worlds

From a VR language point of view, the first version of VRML (VRML 1.0) allows for the creation of virtual worlds with limited interactive behavior by specifying some standards for rendering, networking, limited touch, and vision. The virtual worlds created by VRML can contain objects which have hyperlinks to other worlds, HTML documents or other valid MIME types.

As mentioned earlier, the current VRML format is a subset of the OpenInventor File Format from silicon Graphics plus extensions to support networking. A virtual world can be simply defined by specifying its objects. The objects are defined in terms of their shape (sphere, cube, surfaces...), size, position, material and color. In addition, cameras can be defined as if the user is looking at the scene through that camera. Also light sources and their position are specified.

For example, let us present the following scene in the VRML format:

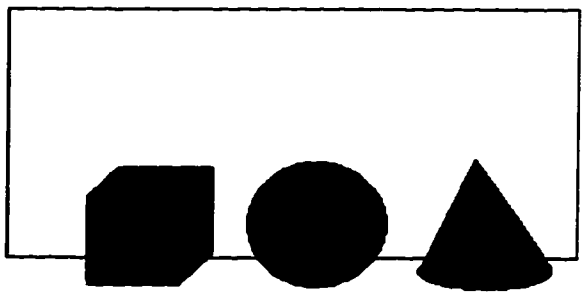


Figure 3. An example 3D scene

This scene basically consists of a red cube, a green sphere and a blue cone in front of a white wall. Here is the corresponding VRML 1.0 description for the above scene:

```
#VRML V1.0 ascii
Separator {
        DirectionalLight {                  # scene lighting
                direction -1 0 -3
                intensity 2
        }
```

```
        PerspectiveCamera {                     # camera looking at scene
                position   0 0 40
                orientation 0 0 1 0
        }
        Separator {
                Coordinate3 { point[-10 -10 5,
                                    10 -10 5,
                                    10 10 5,
                                    -10 10 5] }         # vertices of the wall

                Rotation { rotation 0 0.000001 0 3.1415927 }

                IndexedFaceSet{    coordIndex [ 3,2,1,0,-1] }  # connection matrix for the vertices
        }
        Separator {                                     # green sphere

                Translation { translation 5 -5 0 }

                Material {diffuseColor 0 1 0 }          # color in RGB

                Sphere { radius 1.5 }
        }
        Separator {                                     # red cube

                Transform{rotation 1 1 -1 1.0  translation 1 -5 0}

                Material { diffuseColor 1 0 0 }

                Cube {width 2.2  height 2.2  depth 2.2}
        }
        Separator {                                     # blue cone

                Transform {rotation -1 1 -1 -0.8  translation 8 -4.5 0}

                Material { diffuseColor 0 0 1 }

                Cone {bottomRadius 1.2  height 3}
        }
}
```

Here is a snapshot of the view generated from this description:

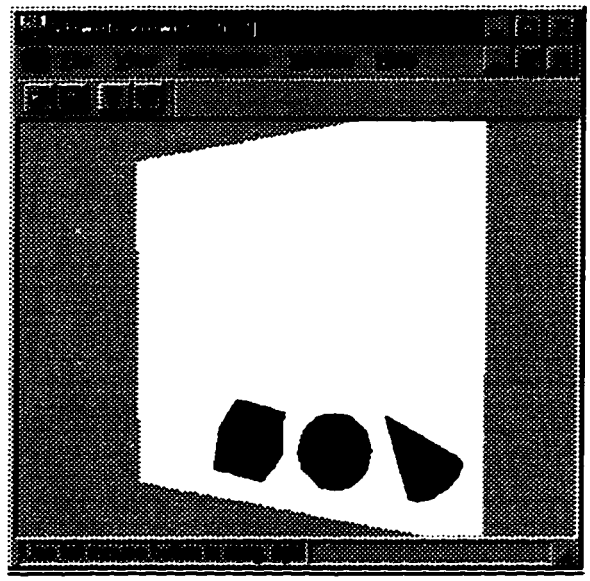


Figure 4. View generated from the scene in figure1 by a browser

Using the mouse, a user can navigate in the above scene and view the objects from different angles and distances. Also, any object can be associated with a hyperlink such that by clicking on that object, the document pointed to by the hyperlink is loaded into the appropriate browser. Hence, as we can see VRML 1.0 has standards for rendering, very limited interaction, and some networking.

In order to create or use VRML files, some software tools are needed. Software applications that allow the users to view VRML files are usually available for free from the Internet. On the other hand, the modeling tools are usually commercial.

2.2.4 VRML Tools

VRML Browsers

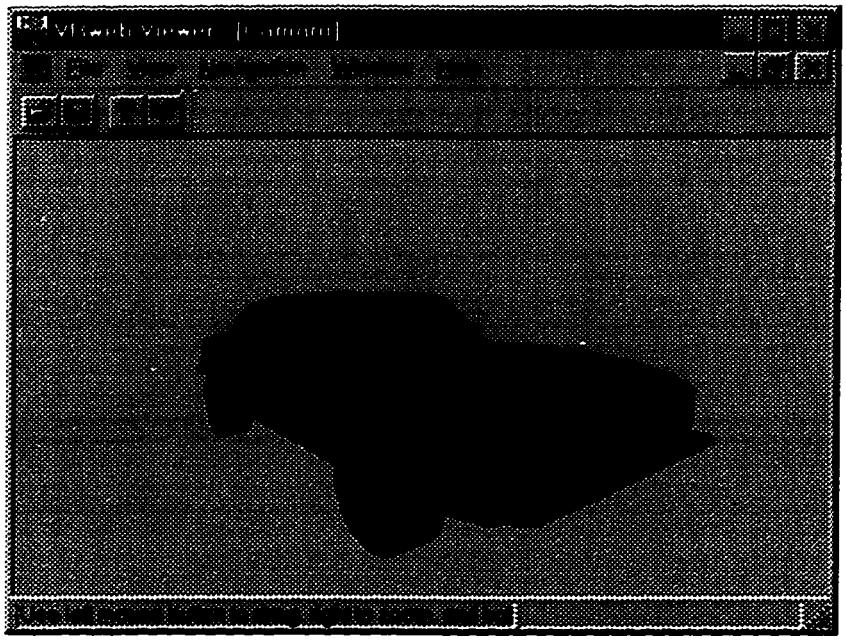


Figure 5. Snapshot of the VRWeb Browser

VRML-Browsers are software tools that generate views from a VRML file. You would

need a VRML-browser to "see" a VRML file as you would need an HTML-browser (like Netscape) to view HTML files.

VRML Modelers

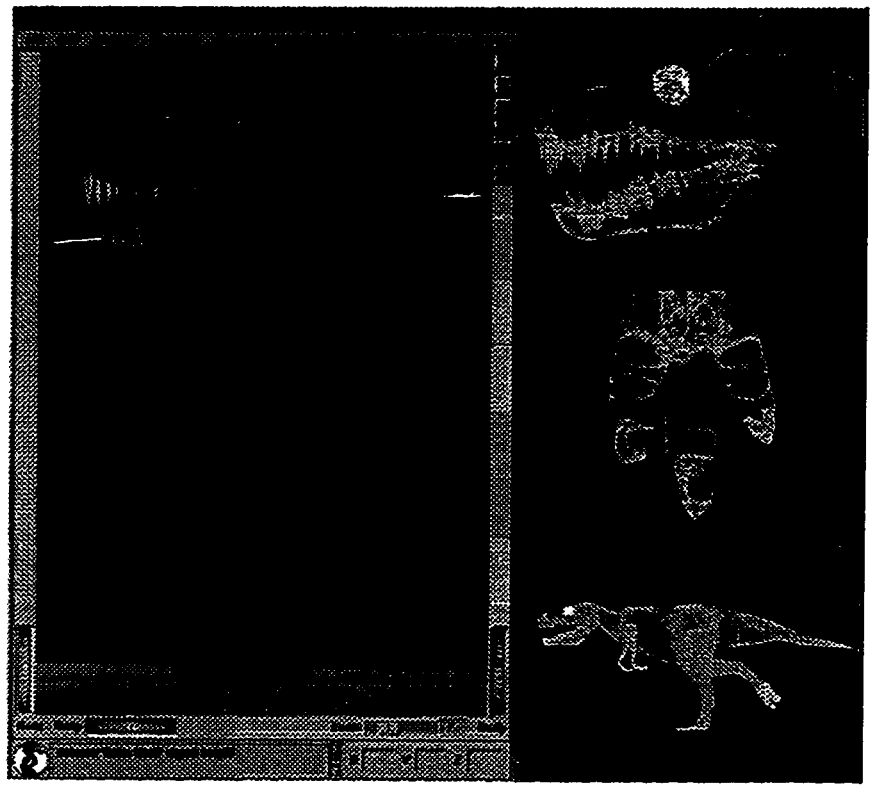


Figure 6. Snapshot of a Modeler

It is extremely difficult, if not impossible, to directly write the VRML description for highly complex shapes (such as the above scene); therefore, there's a need for VRML authoring tools or modelers.

A modeler is used to create three-dimensional scenes or objects. A user can simply draw a scene and the modeler would export it to VRML format. Most modelers also import 3D objects written in other standards such as OFF (Object File Format), Softimage 3D, IBM's Visualization Data Explorer (DX), and many more.

2.2.5 Moving Worlds: VRML 2.0

As we saw, VRML 1.0 defined standards for rendering, very limited interaction, and networking. In order to enhance the capabilities of VRML in terms of a standard for virtual reality as discussed in 2.1, many proposals were submitted. Among these, Moving Worlds by Silicon Graphics [23] was accepted for VRML 2.0.

Moving Worlds furthers the capabilities of VRML 1.0 by introducing the following:

- a scripting language binding in Java and JavaScript;
- events that can be routed from one object to another;
- sound;
- time, motion, click, and other types of sensors.

Java is a platform-independent objet-oriented programming language. We will see more about Java in chapter 3. *JavaScript* is Netscape corporation's extension to HTML implemented in the Java language. JavaScript is a programmable API that allows cross-platform scripting of events, objects, and actions [11][2].

If implemented completely, VRML 2.0 comes very close to a rich VR standard. However, currently there are very few browsers available for VRML 2.0 and none of them completely meet the specifications for VRML 2.0. [22]. Nevertheless, it must be noted that the final specification for Moving Worlds was released on August 4, 1996 [23] and that it is a very new standard. In time, more VRML 2.0-compliant browsers are expected to emerge.

So, VRML can be used as a 3D standard for the tele-learning system. In the following chapter, the issues of a standard for user-interface are addressed.

Chapter 3. Java: Applications through the World Wide Web

3.1 Introduction

As pointed out in the introduction, there is a need for an interface in order to present the multimedia applications such as the whiteboard, document viewer, chat session, and 3D viewer to the user. We also mentioned that it was necessary to use the existing standards for the development of the tele-learning applications in order to make it accessible to the majority of users on the Internet. As a result, it was decided to use the most basic interface to the Internet: a Web browser.

Basically, a Web browser is a window to the Internet that can present various multimedia documents such as text and images in an orderly fashion. A *Web page*, displayed by a Web browser, is a document that contains advanced text, graphics, background templates, and hyperlinks to other documents. The language in which Web pages are created is the Hyper Text Markup Language, abbreviated as HTML. Analogous to VRML, which is a language for presenting 3D data, HTML is a language for presenting 2D data. Practically all Internet users are equipped with an Internet browser. It is by far the most widely used interface to the Internet.

So, documents containing text and images can be presented with HTML using an Internet browser. But how about the applications comprising the virtual classroom: the whiteboard, the chat session, and the 3D browser?

It turns out, there are three main ways to run applications through an Internet browser: using *helper applications*, *plug-ins*, and *applets*.

A helper application is basically a regular platform-dependent application running on a workstation. To use a helper application, the user has to first download the application for his/her specific platform, then install the application on the workstation and make sure it is running properly, and finally inform the Web browser about the application by configuring some parameters in the Web browser. As a result, every time the browser encounters a document that is intended for a helper application, the browser fetches the document, invokes the appropriate helper application and passes the document to the helper application.

Plug-ins take the idea of helper applications one step further. A plug-in application uses the Web browser as its interface and displays its documents inside the browser. As with the helper applications, plug-ins are platform-dependent and have to be downloaded and permanently installed on the workstations.

Applets introduce a different approach for running applications in a browser. An applet is an application that is embedded in a Web page. It is automatically downloaded when the browser encounters the applet's Web page and it is erased when the Web browser's operation is terminated. Furthermore, applets not only use the browser to display their activities, but also run inside the browser. The latter property is the most important feature of an applet; since the applet runs in the browser it becomes platform independent. In other words, running applications becomes the question of whether or not a browser can run applets and not the question of the underlying operating system or platform.

Currently, the language used to write applets is Java. A browser that can run Java applets is referred to as a Java-enabled browser. Today, more than 90% of Web browsers being used are Java enabled. More than 80% of users use the Netscape Navigator browser which

is available for a variety of platforms. Also, close to 10% of users use the Internet Explorer browser by Microsoft which runs on the Microsoft Windows platform. Both these browsers are Java enabled. In addition, Sun Microsystems' Hot Java browser, which is a browser written in the Java language, also supports Java applets.

Other than the availability of Java enabled browsers, there are more indications that Java is becoming a standard for Internet programming:

WebTV, a newly developed device from WebTV Networks, is a set-top box that enables web-browsing with an ordinary television and a phone line; no computers needed. It has its own proprietary software and web browser, which offers most of the features found in Netscape Navigator and Internet Explorer including Java. Users can subscribe to WebTV's online service, which includes an e-mail address and banking services, at a flat rate of $19.95 per month for unlimited usage [24].

Another brand new network product is the *IBM Network Station*, a network terminal that provides network access over Internet using Ethernet and Token Ring connections. The main advantage of the Network Station over the non-programmable "dumb" terminals is in offering graphical interfaces, Java programming capability, and a browser for connection to the Internet, corporate intranets and server networks. It offers plug-and-play simplicity and an intuitive Windows-style graphical interface, but is managed through a server network. This reduces hardware, software and management expense by 50 to 75 percent over traditional personal computers. Here are a few statistics from IBM's Network Station press release [8]:

- 22 percent of all Internet access devices will be non-PC machines by the year 2000, according to International Data Corporation;

- seventy percent of corporate executives say they will use the Internet for business transactions by 1997, according to a Forrester Research Incorporated survey. The study also states that sixty percent of the two million new Internet users each month are from the business world;
- More than 35 million non-programmable terminals are in operation worldwide, according to IBM estimates.

As we can see, many of the future network devices will run Java. These devices are not just computers but simple devices such as televisions, terminals, cellular phones, and so on.

In short, Java has certain advantages over current languages that make it superior for Internet programming. Java applets posses unique features like:

- no downloading and/or installing of any special software if used with a Java-enabled browser or device;
- platform-independence;
- supported by more than 90% of current Web browsers and many of the future network computers and devices, meaning accessibility to a large user population.

Because of the above features, specially the platform-independence and accessibility properties which are very important for this project as elaborated earlier in the introduction, it was decided to use Java applets as the standard for writing tele-learning applications. This choice creates both conveniences and challenges which are discussed in the next section.

3.2 Java and Its Features

3.2.1 What Is Java?

Sun Microsystems, the company that created Java, describes Java as follows:

"Java is a simple, object oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multi-threaded, and dynamic language."

Although the above statement might seem like an advertisement's chain of buzzwords, it truly describes the characteristics of the Java language. Let us briefly examine some of the above features.

Simple: Java is simple in many aspects but mainly because it is easy and quick to learn compared to other languages. It looks familiar to C and C++ programmers even though its language constructors are much smaller. Furthermore, there are no pointers in Java which is one of its most popular features compared to C and C++ as pointers are one of the most bug-prone aspects of C and C++. In addition, Java programmers don't have to worry about garbage collection as it is automatically implemented by the run-time.

Object-Oriented: Unlike C++ which was an extension of C, Java was designed to be object-oriented from the beginning. Java comes with an extensive set of classes, presented in different packages. For instance, Java contains classes which create graphical user interface components (GUI), classes that handle I/O, classes that handle networking, and so on.

Distributed: Java is designed to run applications on networks. It provides the programmer with classes for network connectivity, including sockets.

Interpreted: instead of producing native machine code, which is what C and C++ produce as executables, the Java compiler produces *byte-codes*. A Java interpreter is used to actually run the byte-code. A Java program can run on any platform that has a Java interpreter and run-time system, known together as the Java *virtual machine*. Java enabled browsers implement a Java virtual machine. What makes Java perfect for the Internet is the extremely small size of the compiled byte-codes; it takes very little time to download a Java applet compared to its corresponding C or C++ application.

Multithreaded: Programmers know that writing code in C and C++ that deals with multiple threads can be very frustrating. It is hard to maintain concurrency, and writing code to handle locks on certain routines and variables can result into deadlock situations. Java offers built-in support for handling threads, including concurrency and consistency.

It is interesting to note that in the Java language, applications and applets themselves are objects. A Java "program" is a class that is either an *extension* of another class, an *implementation* of one or more *interfaces*, an extension of a class and implementation of one or more interfaces, or a brand new class.

The difference between a class and an interface is that all methods of an interface are abstract: they are not defined and are implementation-specific.

To "run" a Java application, the application's class must be instantiated; i.e., an object of that class must be created in the Java virtual machine. To run a Java applet, the browser must be told to fetch the applet's corresponding class. This is done by embedding the applet into an HTML document by using a format similar to the following:

```
<applet code=MyApplet.class width=300 height=300>
<param name="file" value="image.gif">
<param name="repeat" value="3">
</applet>
```

When the browser encounters the above code in an HTML file, it will fetch the class MyApplet from the same location as the HTML file and will run the applet in a 300×300 pixel^2 area inside the browser. In addition, the two parameters, *file* and *repeat*, will be initialized in the applet with the values *image.gif* and *3*, respectively. In other words, parameters are a way to initialize applet's contents from the HTML file; they are analogous to command-line options in applications.

Let us now examine some of the classes of Java that are of interest to this research.

3.2.2 Some Advanced Features of Java

The following classes are only some of the classes of some of the packages of the Java language. They are briefly described here because they are core to the tele-learning software engine. For more elaborate description of Java packages, please see references [5] and [13].

Networking

The `java.net` package contains classes that are relevant to networking. These classes include:

- `java.net.ServerSocket`: used by servers to listen for connection requests from clients;
- `java.net.Socket`: implements a socket for interprocess communication over the network;

A *socket* is a module for accomplishing inter-process communication (IPC); a socket is used to allow one process to speak to another via a network, very much like the telephone

is used to allow one person to speak to another. When two entities on separate workstations want to communicate, a socket is established to handle the communication. The entities then use the socket's methods and variables to obtain access to network functions such as sending and receiving data.

Sockets are created for specific *ports*. A port can be thought of as an address for a specific application that is running on a workstation. The port is needed since there might be more than one application running on the same machine at the same time that requires the use of network connections; ports are used to distinguish which data incoming from the network (or outgoing into the network) belong to which application.

Input/Output (I/O)

The `java.io` package contains classes that deal with different I/O streams, including:

- `java.io.InputStream`: provides basic input methods for reading raw data;
- `java.io.OutputStream`: provides basic output methods for writing raw data;
- `java.io.DataInputStream`: provides input methods for reading Strings and primitive data types[4];
- `java.io.DataOutputStream`: provides output methods for writing Strings and primitive data types;
- `java.io.PipedInputStream`: provides the input half of a pipe used to communicate between threads on the same machine;

[4] Premitive data types are: boolean, character, byte, short integers, integer, long integer, floating point numbers, and double precision floating point numbers.

- **`java.io.PipedOutputStream`**: provides the output half of a pipe used to communicate between threads on the same machine.

A *pipe* is a stream that carries data between two modules or threads that are running on the same Java virtual machine.

Although all of the above classes provide different methods for reading and writing their corresponding data; they have no methods for sending or receiving data through networks. However, a socket object can return an InputStream object and an OutputStream object . These objects can be further abstracted into DataInputStream and DataOutputStream objects for more convenient I/O operations. Using the read and write methods of these objects, data can be exchanged across the network.

Hence, the procedure of sending data through the network between a server and a client becomes similar to the following:

Server:

1. start a **`ServerSocket`** object on a predetermined port and wait for connection requests from clients;
2. when an incoming connection request is received, accept the connection and return a **`Socket`** object of that connection to the object that handles the data (the server);
3. the server gets the **`InputStream`** and **`OutputStream`** objects from the **`Socket`** object, further enhancing them to **`DataInputStream`** and **`DataOutputStream`** if necessary;

4. now, data sent by the client can be read from the `InputStream` (or `DataInputStream`) object and data to be sent to the client can be written to the `OutputStream` (or `DataOutputStream`) object.

The following diagram illustrates the above procedure:

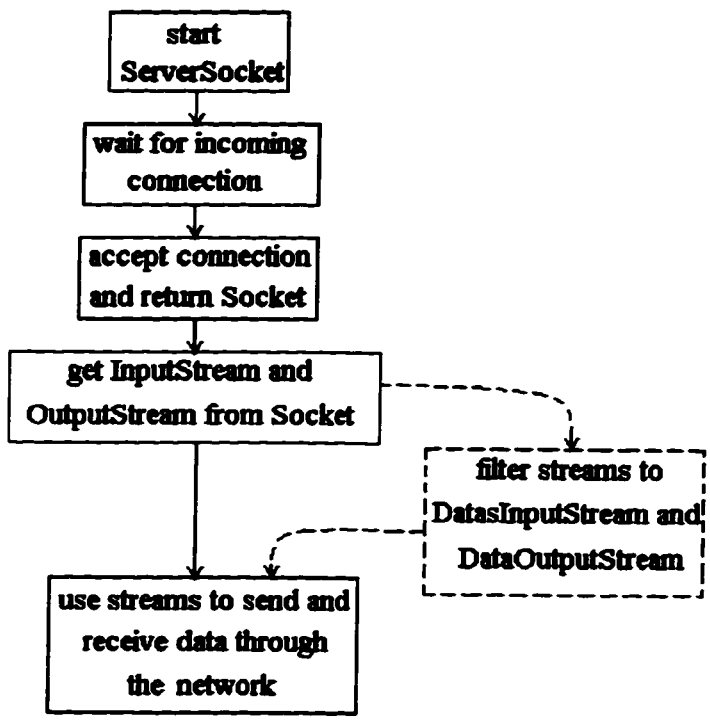


Figure 7. Exchanging information through network sockets: server operations

Client:

1. create a `Socket` object, requesting connection to a predetermined port on a known machine;
2. get `InputStream` and `OutputStream` objects from the Socket object, further enhancing them to `DataInputStream` and `DataOutputStream` if necessary;

3. data sent by the server can now be read from the `InputStream` (or `DataInputStream`) object and data to be sent to the server can be written to the `OutputStream` (or `DataOutputStream`) object.

The following diagram illustrates the above procedure:

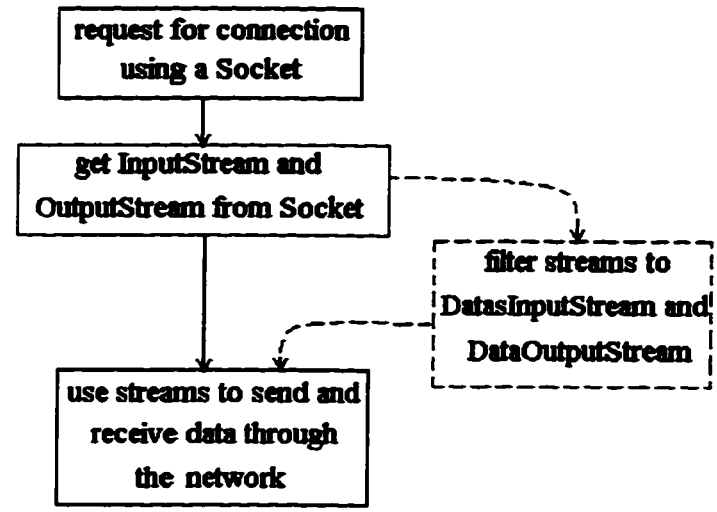


Figure 8. Exchanging information through network sockets: client operations

Graphical User Interface

The `java.awt` package is the Abstract Windowing Toolkit. This package contains numerous classes for implementing graphics including colors, fonts, and polygons; windowing components including GUI components such as buttons, menus, lists, and dialog boxes; and finally layout manager for controlling the layout or mapping of the components into their container.

No specific classes of this package will be described like the `java.net` and `java.io` classes were described because of the large number of `java.awt` classes. Specific classes will be discussed as they come along.

Multi-threading

The `java.lang.Thread` object provides basic means of multi-threading in Java with the following methods:

- `start()`: starts the thread;
- `stop()`: stops the thread;
- `run()`: the body of a thread;
- `suspend()`: temporarily halts a thread;
- `resume()`: resumes a suspended thread;
- `sleep()`: suspends the thread for a specified amount of time.

This class itself is an implementation of the *Runnable* interface which contains the abstract `run()` methods. In addition, the Thread class has methods that provide mans of access to the threads priority.

A thread can be written as an extension to the Thread class, or an implementation of the Runnable interface. Multi-threading can be done by instantiating many thread classes.

3.3 Applets and Their Restrictions

As mentioned earlier, using applets causes both conveniences and challenges. The main advantage of using applets is that once an applet has been written, it can run in any Java-enabled browser on any platform. Other advantages are the result of using Java as a programming language which were discussed in 3.2.

The first disadvantage of using applets is speed. Although speed of applets is more than adequate to run interactive GUI and network-based applications, since byte-codes have to be interpreted by the Java virtual machine before running, Java applets are considerably

slower than their corresponding C or C++ applications. However, Java designers are working on *just-in-time (JIT)* compilers that translate byte-codes into machine code for a particular CPU at run-time. Preliminary JITs have started to appear for the Internet Explorer [12] and the Netscape browser.

The main challenge caused by using applets is overcoming their security restrictions. Since applets are loaded over a network, the only way to make sure that they do not perform any malicious action, like deleting system files and sending fake e-mails, is to run them in a very limited environment. While designing an applet, the designer must keep certain restrictions in mind. There are many applet restrictions. For this research, it must be realized that applets are not allowed to:

- read files on the local system;
- write files on the local system;
- delete files on the local system;
- rename files on the local system;
- create a directory on the local system;
- list directory content;
- check for the existence of a file;
- obtain the type, size, or modification time of a file;
- create a network connection to any computer other than the one from which the applet was loaded;
- listen for or accept network connections on any port of the local system;
- obtain user's name or home directory name;

- define any system properties;
- invoke any program on the local system.

As one can see, these restrictions are rather severe when approached by the conventional C or C++ programming perspective. Therefore, it is evident that programming applets must be with the mentality which is different from that of the conventional programming. For instance, it is obvious that if applets are ever to communicate with each other, it has to be through a central-server architecture as applets can only make network connections to the machine from which they were downloaded.

In general, the approach to designing advanced applet-based systems, such as the tele-learning system, must be that of an object-oriented, distributed, and client-server architecture. This was the approach used for designing the tele-learning architecture as completely described in the following chapter.

Chapter 4. Design of the Multiuser Client-Server Model

From design point of view, the main characteristic of the tele-learning system is that it should be simple, generic, and robust. The system should neither be too complex nor too specific as those will prevent integration of the system with other systems, as well as limit the evolution of the system and make it useless in the long run. On the other hand, it should not be too simple either as that will cause a lot of effort and work for the future developers of the system to enhance its features. Currently, most existing approaches to shared environments offer some sort of primitive distribution mechanism such as asynchronous message passing which gives little help to the developer because all distribution and synchronization issues must be dealt with explicitly[18]. The tele-learning system must provide functionality which is the common-denominator of functionalities required by all collaborative applications, plus some advanced features. With that in mind, the specifications of the system are described next.

4.1 General Specifications

As mentioned in 3.2, a centralized-server architecture is inevitable because applets can only communicate to the computer from which they were downloaded. Any message-passing as well as other types of communication must be done through the server. Hence, it becomes apparent that the server must be able to handle asynchronous message-passing amongst applets.

Since the environment is multi-user, issues such as *access control* and *consistency* also become a problem. In this thesis, access control is referred to the action of assuring that only one user has access to a shared object at a time and that no two or more users can

modify an object at the exact same time. Consistency is referred to the state that all users are simultaneously presented with the same data and objets.

All of the above are common to collaborative applications; though not all applications require both consistency and access control. In addition, the server should also be able to monitor clients' states (active, silent, dead) and to exchange signals with specific clients. So, the features that the server must provide are:

- consistency
- access control
- asynchronous data passing
- monitoring of clients
- signaling ability between client and server

The first three features are justified based on the system's needs. Monitoring of clients becomes necessary for security reasons. It should be possible for the server's administrator to see who is logged into the system, who is accessing objects, and what state individual clients are in. Signaling ability between a client and the server is also necessary for various reasons such as a client requesting access to a shared object an so forth; there is a need for the client to talk to the server privately.

The client must inherently posses the following features:

- automatic network connection to the server
- a client thread to handle incoming messages in real-time
- access monitor
- automatic signaling of the client's status to the server

- dialog with the user

As a built-in feature, the applet, once downloaded into the browser, should be able to automatically establish connections to the server from which it was downloaded from. Also, the applet should fork at least one thread which is responsible for real-time handling of incoming data and messages. It is important that this task be performed as a separate thread since the applet itself is interacting with the user in real-time. This results in instantaneous response to user interactions as well as incoming data and makes the system more real-time.

However, it might well happen that the incoming message causes the applet to perform a task which is conflicting with current user interaction. For example, in a shared 3D object viewer, the incoming message might indicate that the object should be rotated to the right by 90° whereas the user might want to move the object forward by 2 meters. That's why there is a need for an access monitor at the client. The client must provide a means of requesting access to a shared object as well as releasing an accessed shared object.

Also, the client must automatically inform the server of the client's state. For example, when the user minimizes the browser the applets become inaccessible to the user for as long as the browser is minimized. In this situation, the client applets should inform the server that they are silent and not active. When the user quits the applets or the browser, the client applets must inform the server to close down the connection and free network resources for other clients.

Finally, the client must have methods to show messages to the user in order to inform the user about some actions such as downloading file and parsing 3D object, as well as error or warning messages.

The proposed client-server architecture provides an infrastructure for developers to create collaborative applications without having to deal with the main issues of concern for collaborative environments. Programmers can build shared applications just by writing the code for what the application is supposed to do without ever worrying about issues such as communications to other clients, access control, consistency and so on. These issues become abstract to the user since they are automatically performed by the tele-learning system.

Consequently, the goal here becomes the creation of a shared Server class and a shared Client class that forms the basis for the development of shared applications. Developers can then extend these generic classes to write code for their specific application. The architecture for the Server and Client classes is presented next.

4.2 Proposed Architecture

In order to achieve maximum real-time behavior, there is one Server object launched for each application. If there is only one server serving all clients of all applications, that server will be severely hit by performance problems. At the other extreme, using one server for each client creates to many servers and overhead on the machine running the server. Therefore, it makes sense to have one server for each application, serving all the clients for that application. Each individual server can then launch small threads for each of its clients to provide even further real-time behavior.

The architecture involves both the layout and the operation of the tele-learning system.

4.2.1 Layout

SERVER

The general layout of the server is shown in the following figure:

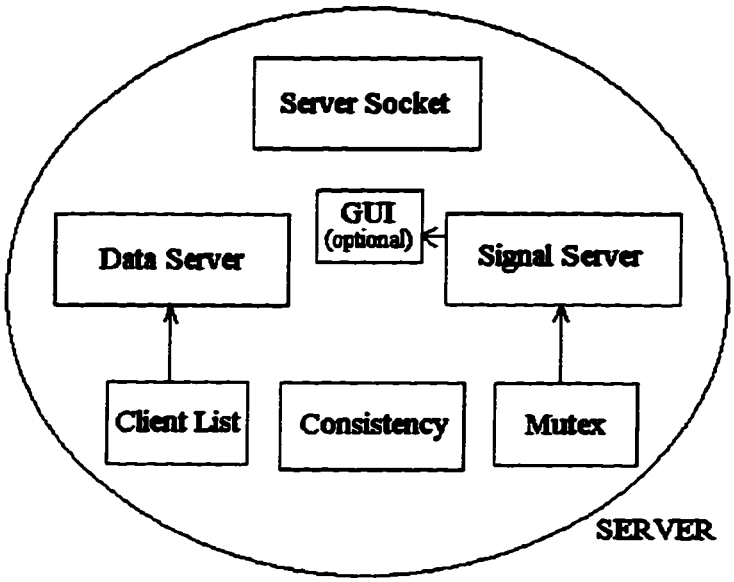


Figure 9. Context diagram of the Server

As it can be seen, the server consists of the following objects:

- **`ServerSocket`:** responsible for accepting connections from clients;
- **DataServer:** responsible for asynchronous data passing between clients;
- **SignalServer:** responsible for handling and exchanging signals with a specific client;
- **Consistency:** monitors interactions with shared applications and can be accessed to find out the current state of a shared object;
- **Mutex:** consists of a semaphore and its operating methods that can be used to attain or release lock on a shared object;
- **ClientList:** an array of **`OutputStream`** objects that corresponds to output streams to all of the clients.

Except for the ClientList and the Mutex objects, all the above objects are real-time threads. This ensures maximum real-time behavior from the server. The DataServer, SignalServer, and Consistency objects are forked after the ServerSocket has accepted a connection. The forking procedure is explained later in details.

Note that the SignalServer and the DataServer should not only run as separate threads, but also use separate Sockets to communicate with the client. This separates the job of signaling and data passing; the system can be thought of having a *Data Channel* and a *Signaling Channel.* The advantage of this, other than a highly real-time performance, is that data and signals can be sent independently and simultaneously.

There is also GUI-enabled version of the Server class which in addition to the above objects, also contains a GUI object. This object is used to visually display the name, status and object-accessing of clients to the server administrator[5]. This display could be in form of a table similar to the one shown below:

CLIENT	STATUS	ACCESS
machine1.ottawa.ca	active	X
machine2.toronto.ca	silent	

Figure 10. Sample monitoring interface

The reason the GUI object is not included in the standard Server class is that servers usually run at all times, even when server administrators are not logged into the server machine. GUI interfaces are killed upon log-out of users, resulting in interruption or

[5] In this thesis, server administrator is referred to the person that is responsible for the operation of the server.

destruction of their corresponding processes, in this case the server. Therefore, the GUI-enabled version of the server is used only for short tele-learning sessions or for machines that are dedicated as servers, meaning the supervisor is always logged in.

Next, the client architecture is presented.

CLIENT

The general layout of the client is shown in the following figure:

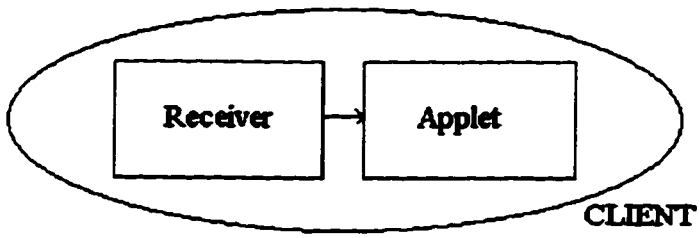


Figure 11. Context diagram of the Client

The client basically consists of the these objects:

- **Receiver:** responsible for receiving and handling the incoming data in real-time ;
- **Applet:** the applet body itself.

The Receiver is a thread; it always listens for incoming data and performs appropriate actions upon receiving data. This object receives data from the DataServer object of the Server class. The Applet is an extension to the `java.applet.Applet` class which is the regular applet class used in Java. The Applet class is responsible for automatic network connection establishment, status signaling, and access monitor. It automatically communicates with the ServerSocket and the SignalServer objects of the Server class.

4.2.2 Operation

CLIENT

Once downloaded into the browser, the Client applet establishes a Socket connection to the server. From the Socket, it gets I/O Streams for Data channel and, if required, I/O streams for Signaling channel. Data channel will then be filtered from InputStream and OutputStream to DataInputStream and a DataOutputStream to facilitate I/O operation for primitive data types. As mentioned before, the InputStream and OutputStream objects only support reading and writing of raw data. The flowchart below shows the operation of the Client:

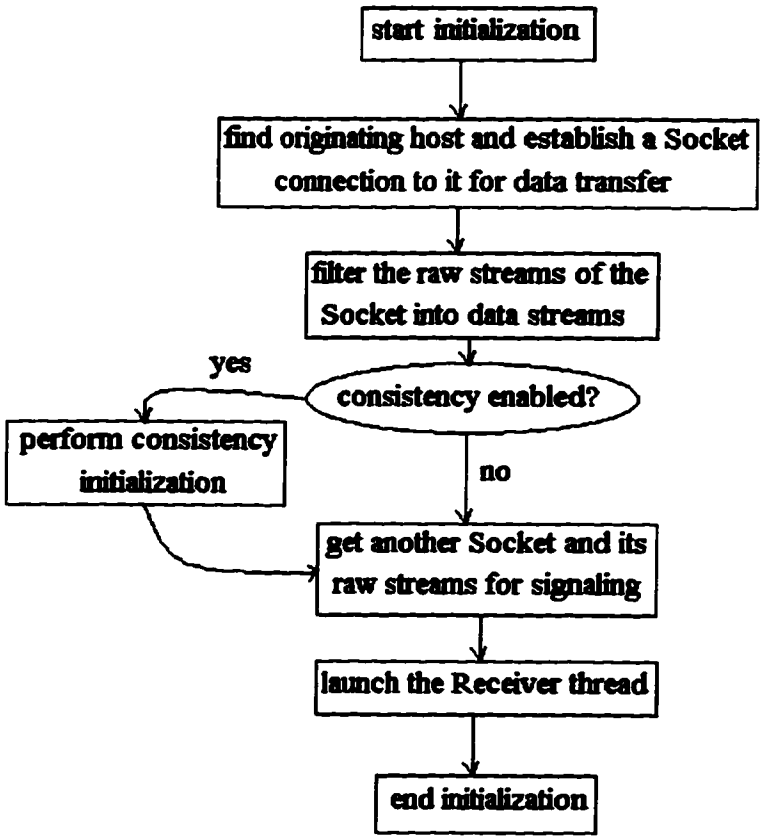


Figure 12. Applet initialization automatically performed by the Client

For applications that require consistency, the Client applet also performs an application-dependent initialization for consistency. This means that it will receive consistency data from the server in order to adjust itself with the current session in progress. Since the consistency data is application-dependent, an abstract initialization method for the Client class is provided that has to be defined by the programmer.

Note that all applications do not require consistency. It would be a waste of resources to provide consistency (on the server and on the client) for applications that do not need it. This is why consistency is supported as an option. We will see more about consistency when the Server operation is described.

The Client also has an abstract run() method. This is the Receiver part of the Client. In this method, programmers must define what kind of data the applet should be expecting, and what the applet should do with those data. For example, in the case of the shared whiteboard, the Receiver should expect four integers representing coordinates of the two end points of a line, and should draw a line using those numbers.

Moreover, the application programmer must keep in mind that data needs to be sent through the Data channel every time the user interacts with the applet. For instance, when the user draws a line in the whiteboard application, the client should send four integers to the server, representing the coordinates of the two endpoints of the line. This data is sent to the DataServer object of the server, which than relays this data to all other clients and the Consistency object, if applicable.

The pre-determined way in which applications send and receive data amongst themselves is labeled *Messaging Scheme*. When a user interacts with an applet, the specific interaction is mapped into a message, and the message is sent to other applets. The receiving applets,

translate that message into an action using the same messaging scheme and perform that action. For instance let's assume that in a 3D-viewer applet, the messaging scheme is similar to the following:

0: rotate
1: move
2: zoom
3: get 3D file

When one of the users rotates the 3D object by 20°, the 3D viewer applet maps this action into 0-20, meaning rotate by 20°. This message is sent to other applets where it is translated into the appropriate action using the same messaging scheme.

Hence, it becomes the responsibility of the programmer to build a messaging scheme for a particular shared applet. This scheme must be implemented in the run() method of the Client class for receiving messages, as well as in the *event-handling methods* for sending messages. Event-handling methods are methods such as mouseUp(), mouseDown(), mouseDrag(), and so forth.

Although the messaging scheme is the fastest way of reflecting interaction from one client to the others, it is not the most convenient method for the programmer. The more complex an application is, the larger its messaging scheme becomes. The easiest way, but not fastest in terms of performance, is to use some technique similar to *Remote Procedure Call (RPC)* in C and C++. Using this technique, a client can directly invoke other client's methods. For instance in the above example, if the 3D viewer application has a method called **rotate**(float theta), an initiating client has to remotely call **rotate**(20) on all other clients to reflect its user's interaction.

Presently, Java does not support RPC. However, a similar technique called *Remote Method Invocation (RMI)* is under development for Java. As of the day of writing of this thesis, RMI is not available as part of the core Java packages.

The Client applet also has the following built-in parameters:

TABLE 1. BUILT-IN PARAMETERS FOR THE CLIENT CLASS

Parameter Name	Description	Default Value
name	name of the application	Applet
port	port number on server	-
host	hostname of the server	"applet's host"
signal	signaling option	false
consistency	consistency option	false

name indicates the name of the applet, for example "3D Viewer". The port number is the one the applet uses to connect to the server.

The *host* parameter contains the domain name of the server for example: "mango.genie.uotawa.ca". The question that arises here is that why should there be a parameter for specifying hostname if the applet is able to find the hostname from which it was downloaded from? Specifying hostname causes re-writing of HTML file when the server machine is changed and creates inconvenience.

The answer is that because of security reasons, some computers are protected by a *firewall* and are connected to the Internet through a *proxy server*. A proxy server fetches Internet documents for its clients, meaning that the computers behind the firewall which are not connected to the Internet can have Internet access through the proxy server [1]. The problem occurs because the Web browser of a client that uses a proxy server thinks the applet's host is the proxy server and not the real server. So it must be told specifically

to request connection to the applet's server and not the proxy server. That's why there is need for the host parameter.

The *signal* parameter indicates whether or not a signaling channel is needed. The Server class always offers a signaling channel; however, specific applets might not need to use a signaling channel. If the server is launched with the GUI option to monitor clients, then there is a need for all applets to use a signaling channel to report their state to the server. Otherwise, the only other time applets require signaling is when they need to lock or release a shared object. Therefore, if applets neither report their state to the server nor need to lock a shared object, they won't require a signaling channel. Examples of such applets are chatting applets, or shared whiteboard, or shared HTML viewer to some extent. These applets really don't need to lock their application at any time.

There is also the *consistency* parameter which indicates whether or not the applet should perform any initialization for consistency purposes.

SERVER

The Server can be started with or without one or both of the two options: consistency and access control. This flexibility is provided since different applications require different types of distributed service. Some applications, for example the whiteboard, require consistency but not access control. The whiteboard needs consistency since a new user should be provided with what has already been drawn on the whiteboard; however, access control is not required since it is all right for more than one person to simultaneously draw on the whiteboard. Other types of applications require both consistency and access control; 3D object viewer is an example of this type.

So, depending on the type of the application, the server can provide consistency or access control or both. It would not only be a waste of resources to provide both the above services by default, but also a problem for the applications that don't need them.

The operation of the Server is illustrated in the following figure.

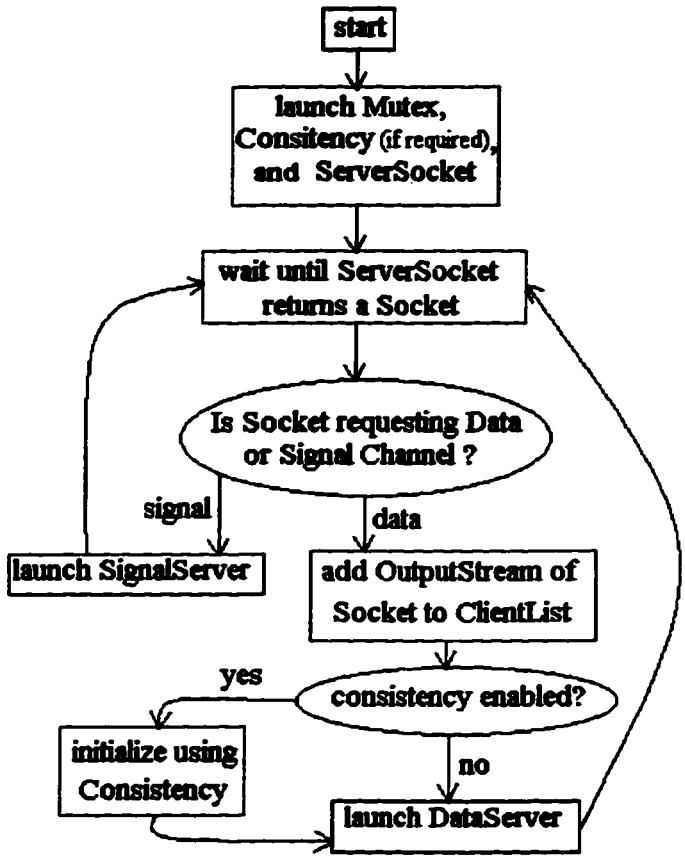


Figure 13. Operation of the Server

The Server starts by launching a Mutex and a ServerSocket object and, if instructed to, a Consistency object. The ServerSocket will listen to connection requests on a specific port and accepts connections if the current number of clients is less than the maximum number of clients allowed. Upon acceptance of a connection, the ServerSocket returns a Socket object representing the client. Depending on whether the client needs that Socket for data channel or for signaling channel, the server takes appropriate action. If the client is

requesting a signaling channel, the Server passes the Socket to SignalServer and goes back to waiting for new requests. In case of a data channel request, the Socket returns an InputStream and an OutputStream for data communication. The OutputStream is added to the ClientList. The Server than performs initialization procedures if consistency is enabled, and launches a DataServer thread passing to it the InputStream returned form the Socket plus the ClientList.

As we can see, there will be a separate DataServer thread running for each client. This will ensure that the message exchanging speed of the Server is maximized as these threads run independently.

DATA SERVER

The following flowchart shows the operation of the DataServer:

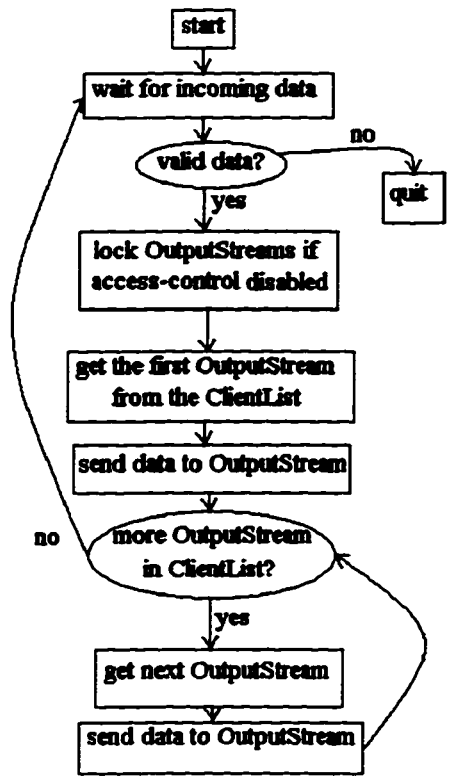


Figure 14. DataServer operation

The DataServer, now running as a separate thread, listens to incoming data from its corresponding client on the InputStream and relays the data to all the OutputStream objects in the ClientList, except for the client itself. At the InputStream level, this data is raw data; a bunch of ones and zeros. The DataServer doesn't know what the data represents: an integer, a String, or an image. This will ensure privacy of data being communicated and more important fast performance of the server. So as we can see, the DataServer accomplishes the task of data-passing.

In the case of access control being disabled, the DataServer locks the OutputStreams in the ClientList before sending any data through them. This action is not performed if access control is enabled. This can be explained as follows:

When there is no access control, every user can unrestrictedly interact with the application. As mentioned before in the client operation section, these interactions are translated by the pre-determined messaging scheme and sent to other clients. The problem occurs when two or more users do something with the application at the same time.

For example, suppose for application X that does not need access control, client1 sends the message "4,23.1,Hi" to its DataServer whereas at the same time client2 sends the message "4,24e9" to its DataServer. These messages *collide* at the Server, producing a new message which is not comprehensible for other clients (see figure 14). The result could be anything from random graphics patterns appearing on the other clients, to Server or DataServer crashes. This problem is prevented if each DataServer locks the OutputStream before sending its message. But as mentioned earlier, the DataServer does not know what data is being transmitted as it only sees bytes. Therefore, a special control byte is sent by the client every time the message is finished. This problem does not occur

for the case where access control is enabled, of course, since in that case only one user at a time is allowed to interact with the application.

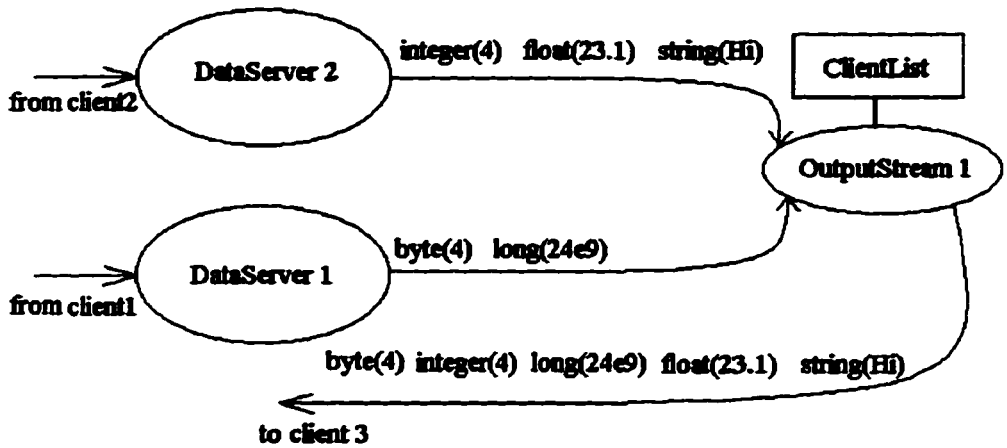


Figure 15. An example of message collision

In addition to the above key operations, the DataServer automatically performs garbage collection. When the DataServer's corresponding client quits, it closes the InputStream and OutputStream of the client and removes the client form the ClientList. It then shuts itself down.

SIGNAL SERVER

The SignalServer, also running as a separate thread for each client, listens to the signals coming from the client and takes appropriate actions. These signals are sent as bytes with different values. The following table summarizes the built-in signals of the SignalServer.

TABLE 2. BUILT-IN SIGNALING OF THE SIGNALSERVER CLASS

Signal	Name	Mutex	Action	Response to Client
0	Mutex_release	locked	release Mutex	-
1	Mutex_lock	locked	-	0
1	Mutex_lock	free	lock Mutex	1
2	silent	-	notify GUI	-
3	active	-	notify GUI	-
4	blink	-	notify GUI	-

The first two signals deal with access control. When the client wants to access a shared object, it sends a lock request (Mutex_lock) to the SignalServer. If Mutex is already locked, the SignalServer denies the client request by sending it back a signal of value zero; otherwise, the Mutex is locked and the client is informed about its request being granted by being sent back a signal of value one.

The last three signals are used in the GUI-enabled version of the server. In this version of the server, each SignalServer is given a *Display* area in the GUI interface. This Display area consists of a *name filed*, a *status field* and an *access field*. The name filed displays the hostname of the client. The status field is used to display the status of the client, while the access filed is used to indicate whether or not the client is accessing the object.

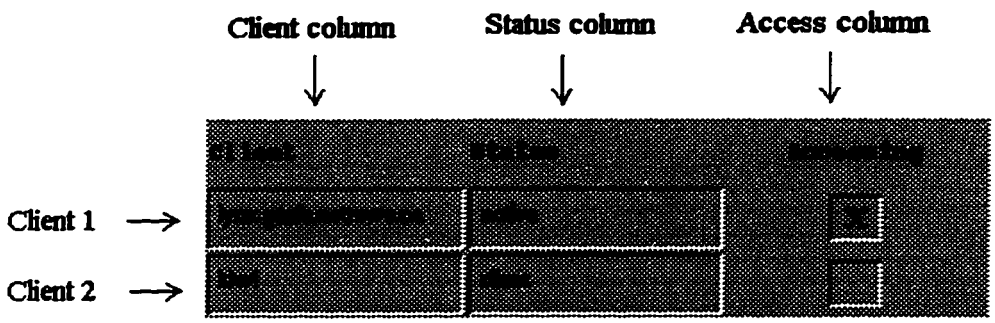


Figure 16. Optional GUI interface of the server for monitoring clients

The silent signal is automatically sent by the client when the applet is not accessible to the user because of the browser having been minimized for example. This is reflected in the status field of the Display area. The active signal is automatically sent by the client to indicate that the user is actively using the applet. This is also reflected in the status field of the Display area. The blink signal can be sent to indicate temporary activity by user. This is done by putting an X mark in the access field of the Display area for 500ms and removing it (blink). Furthermore, in the GUI-enabled version, the X mark is put in the access filed

when a client has locked the object and it is removed upon releasing the object. The SignalServer closes the client Socket when the client has quit. The flowchart below summarizes the operation of the signal server:

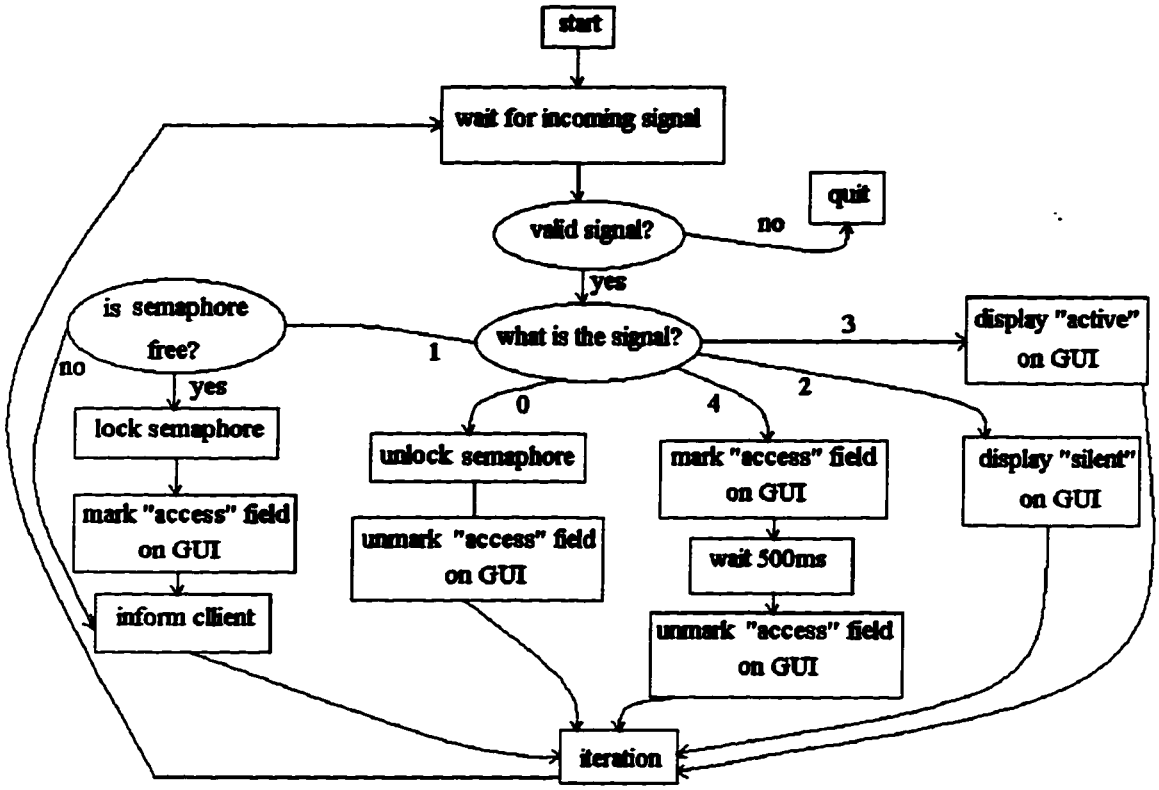


Figure 17. SignalServer Operation

The Mutex object is also of interest here. As described earlier, this object consists of a semaphore of type Boolean plus two operating methods. At the beginning when the Mutex is initialized, the value of the semaphore is set to false, meaning that it is not locked. The object has two methods: read() and set(). The read() method is used to read the value of the semaphore and the set() method is used to set it. The semaphore is locked by setting its value to true. Both these methods are *synchronized*, meaning they will lock the entire Mutex object before performing any action. This will ensure that one client cannot read

the semaphore while another is setting it. Also, in case the semaphore is free, two clients cannot read it as free and lock it at the same time.

CONSISTENCY

The last built-in function of the server and the most complex one, is the Consistency object. The Consistency object is another thread which monitors the state of the shared application. For example, the Consistency object of a 3D browser monitors all the rotations, motions, and coordinates of the 3D object. It simply monitors what the users are doing with the object and keeps the state of the object up-to-date. Hence, it is application-dependent. The consistency initialization as performed by the Server (figure 12) is shown below:

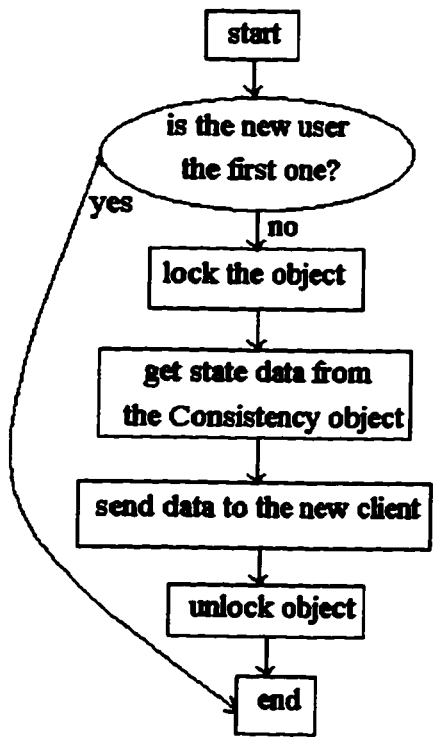


Figure 18. Initialization operation as performed by the server

The main use of the Consistency object is for a new user joining the session in the middle of it and not from the beginning. When a user joins the tele-learning session, it is notified

by the server about whether or not he/she is the first user to enter the session. In case of a first user, there is no consistency problem and initialization is minimum.

However, in case the new user is not the first one, the application is locked, the Consistency object is asked for the data representing the current state of the application, that data is sent to the new client, and the application is unlocked. All of this is performed by the server at the initialization stage, before launching the DataServer.

It is apparent that this initialization is application-dependent. For a 3D viewer, the initialization data represents the present coordinates of the 3D object; whereas for a shared whiteboard, the data represents what's already been drawn on the whiteboard. This forces the Server class to have an abstract initialization method.

Hence, it becomes necessary for the programmer to define two things for the server: the Consistency object, and the initialization method.

The Consistency object is relatively simple to program. It is the same as the target application except it doesn't display anything, receive or send any signals, or send any data. It only receives data and updates itself in the same fashion as the application. For instance, the Consistency object for a shared whiteboard application is the same as the whiteboard application except it draws its images and lines into an Image object without displaying it. That Image is used at the initialization stage to provide consistency for the new user.

If the server is started with the consistency option, the first entry in the ClientList is a `PipedOutputStream` to the Consistency object. This way, when DataServers send incoming data to all clients using the ClientList, they automatically send those data to the Consistency object as well without ever knowing it exists.

The initialization method of the server is also easy to program: get the initialization data from the Consistency object and send it to the new client. For instance, get the Image data from the Consistency object of the shared whiteboard application and send it to the new user.

The following section is an attempt to summarize the above layout and operations.

4.3 Summary

The Server class contains the following classes and methods:

TABLE 3. SERVER CLASS COMPONENTS

Name	Task	Has access to
ServerSocket	listens for and accepts new connections	-
Mutex	contains a semaphore and synchronized methods for reading and setting it	-
Consistency*	monitors the state of the application	-
DataServer	performs asynchronous data passing through the Data channel	ClientList
SignalServer	performs signaling through the Signaling channel	Mutex, GUI
ClientList	contains OutputStreams (PipedOutputStream for Consistency) of al clients	-
GUI*	consists of Display areas to show various client activities	-
initSetup()	obtains data, representing current state of the application, from Consistency and relays it to the application	Mutex, Consistency

* optional

Except for the Mutex and the ClientList class, all other classes are running as separate threads to ensure high performance. There is one DataServer running for each client and, if required, one SignalServer running for each client. The initSetup() method, which is used for consistency initialization, must be defined by the programmer and is application-dependent. In addition, the Consistency object must be separately coded and included in

the Server for applications that require consistency. This object is a simulation of the actual Client applet.

The Client class has the following classes and methods:

TABLE 4. CLIENT CLASS COMPONENTS

Name	Task
dataIn	DataInputStream of the Data channel
dataOut	DataOutputStream of the Data channel
signalIn	InputStream of the Signaling channel
signalOut	OutputStream of the Signaling channel
run()	receives data and handles them
initSetup()	receives initialization data and handles them*

* if consistency is enabled

The initSetup() method, which is used for consistency initialization and communicates to the initSetup() method of the Server, must be defined by the programmer as it is application dependent. Furthermore, a messaging scheme must be implemented in the run() method and the event-handling methods of the applet to enable clients inform each other about their user's interaction.

In addition, the Client class takes the following parameters from the HTML file containing the client applet:

- **name**: name of the application;
- **host**: hostname of the server for clients that access the Internet through a proxy server;
- **port**: predetermined port number on the host corresponding to this application;
- **signal**: indicates whether or not there is a need for signaling channel. Must be set to *true* for application that require consistency;
- **consistency**: indicates whether or not consistency is required.

The following figure is the Open Distributed Processing (ODP) engineering model of the

tele-learning system, illustrating the peer-to-peer communication of methods:

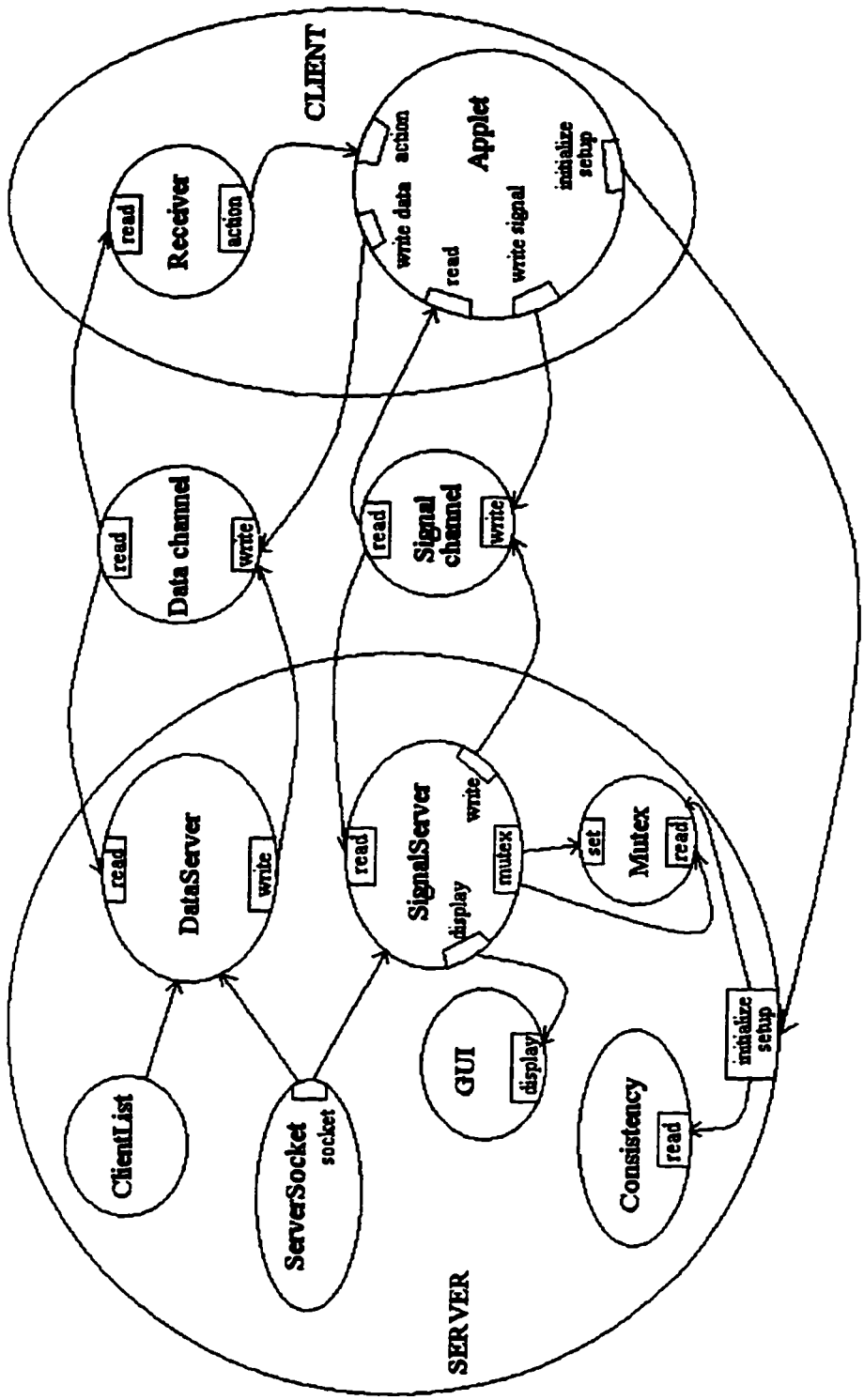


Figure 19. ODP engineering model of the tele-learning system

Using the above architecture, two main classes, Server and Client can be developed. The Client class can be extended to implement a specific shared application. The Server class can be extended to implement the server for that specific application; however, the extension of the Server class is minimal for typical applications and almost zero for applications without consistency requirement. The majority of work, in order to create shared applications, is performed to extend the Client class.

In the next chapter, the implementation of sample applications are presented in order to show the realizability and implementability of the proposed architecture.

Chapter 5. System Implementation

Using the architecture presented in chapter four, including specifications, layout, and operation, generic **Server** and **Client** classes were written using the Java language. Although the Server can be written in any other language, such as C or C++ in order to provide high performance, Java was used for the prototype to make the server platform-independent as well. The following section presents the API of the JETS system.

5.1 JETS Application Program Interface (API)

The API of the JETS sytem is very easy to use. The constructors and public elements have been kept ot a minimum, eventhough the functionalities and features of the system are optimum. The Server class in specific, has a very small API since most of the work is automatically done by the class; there is ver little the programer has to do. The Client class has been given a larger interface since most of the non-generic functionalities are needed at the client application. With this intro, here is the API of JETS:

5.1.1 Server

```
public class Server extends Thread {
        // Public Constructor
        public Server(String name, int port, int numberofClients, boolean consistency,
                        boolean access, boolean gui);

        // Public Instance Objects
        public OutputStream out[];  // clients list
        public PipedOutputStream pipe;  // out[0] in client list (consistency)

        //Public Abstract Methods
        public void initSetup(int outputStreamNumber) ;
}
```

name is the name of the server; for example *VRMLServer. port* is the port number that the server is supposed to listen to. *NumberofClients* indicates the maximum number of clients allowed t access the serve at a time. This is used to prevent overwhelming of the server. *consistency* indicates whether or not the corresponding application requires consistency. If *consistency* is *true,* the server performs the *initSetup()* method; if *consistency* is *false,* the server ignores the *initSetup()* method.

access indicates if access control is enabled or not. This has direct effect on the way the DataServer handles incoming data from clients. *gui* indicates whether monitoring of clients by the server is needed.

out[] is an array of OutputStream objects corresponding to data channels of all clients. This is used by the DataServer to relay incoming data to all clients. If consistency option is enabled, the first entry in the client list, out[0], is a PipedOutputStream object called *pipe.* This pipe can be used to relay data to the Consistency object. So, the Consistency Object will have a PipedInputStream that acts as the receiving end of the information sent by DataServer through the pipe. Just a reminder that the data represents user interaction according to the messaging scheme. The Consistency object uses this data to simulate the application as mentioned before.

The *initsetup()* method is an abstract method that must be defined for each specific server. It should contain the code that performs the consistency initialization operation. It is necessary to define the *initsetup()* method even if the application does not require consistency, in which case the method can be defined empty as follows:

```
public abstract void initSetup(int i) {
}
```

5.1.2 Client

```
public class Client extends Applet implements Runnable {
        // Default Constructor: public Client

        //Public Instance Objects
        public DataInputStream dataIn;
        public DataOutputStream dataOut;
        public InputStream signalIn;
        public OutputStream signalOut;

        //Public Abstract Methods
        public boolean initSetup();
        public void run();

        //Public Instance Methods
        public void destroy();  // Overrides Applet.destroy()
        public void displayStatus(String message);
        public void end();
        public boolean getMutex();
        public void init();
        public void releaseMutex();
        public void start();  // Overrides Applet.start()
        public void stop();  // Overrides Applet.stop()
}
```

Built-in parameters: *name, port ,host, signal, consistency.* (see page 51)

dataIn and *dataOut* are the input and output of the Data Channel; they are used to transfer primitive data types. *signalIn* and *signalOut* are the input and output of the Signal Channel; they are used to transfer signals between the client and the server. They are activated only if the signal parameter is set to *true*.

The *initSetup()* method must contain code for consistency initialization and is executed only if the *consistency* parameter is set to *true*. The method returns *false* if consistency initialization was not successful.

The *run()* method must contain code that implements the Receiver of the client. It should listen and read incoming data and perform appropriate action according to the messaging scheme.

The *destroy()*, *start()*, and *stop()* methods are automatically called by the Web browser running the applet. *destroy()* should contain some garbage collection instructions like closing the sockets. *start()* is called every time the applet is displayed to the user. *stop()* is called every time the applet is hidden from the user, for example when the browser is minimized.

For the applications that do not require access control, *end()* is used to indicate to the Server end of message. When DataServer sees the end command, it will lock the OutputStreams and send the message.

getMutex() checks the status of the semaphore running on the server and requests a lock on the application. It returns *true* if lock is granted, *false* if lock request is denied. *releaseMutex()* is used to unlock an application that is locked. Note that an application can only be unlocked by the client that has locked it.

displayStatus() is used to show error or other messages to the user. It uses the browser's status bar to display theses messages.

5.2 JETS Prototype

The above API was used to develop a prototype of the JETS system. Four applications that form the basis of a virtual classroom were developed. These applications are presented next.

5.2.1 Chat

The Client class was extended to create a chatting applet. Using this applet, a user can send textual messages to other users as well as view other users' messages. In order to distinguish which user is sending which message, all users are asked to enter their name into a text field in the applet. Messages are sent by writing them into the appropriate text field and hitting the ENTER key. In addition, messages sent by other users are displayed in a text area in the applet. Below is a picture of the Chat applet:

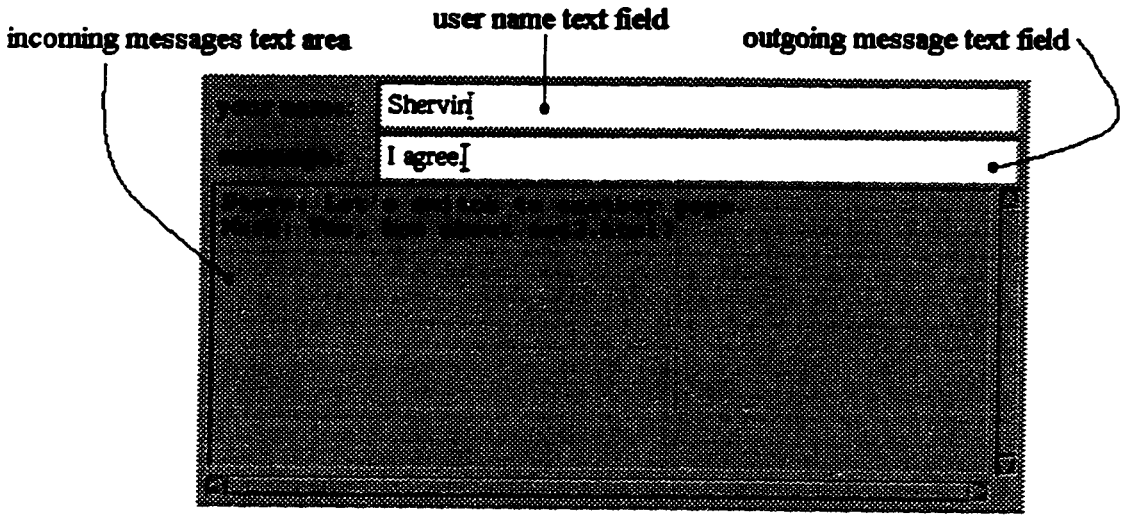


Figure 20. The chat applet

As far as the server for the Chat applet is concerned, the only extension to the Server class was to indicate a port number and a name for the server (Chat Server).

There is no consistency for this applet. Consistency could be used to present a newcomer client with the past messages that have been exchanged; however, at the time of making this applet, no need for such action was felt necessary.

Also, there is no messaging scheme for this applet since only one message is sent: the content of the textual message.

5.2.2 Shared HTML Documents

This applet (called URLFetcher in the code) enables one user to share an HTML document with other users. This is done by the user entering the Universal Resource Locator (URL) of the HTML document into the appropriate text field in the applet (see picture below). The applet then asks the browser to fetch the HTML document and display it in the document frame. All other users will also see the same document as requested by the initiator user.

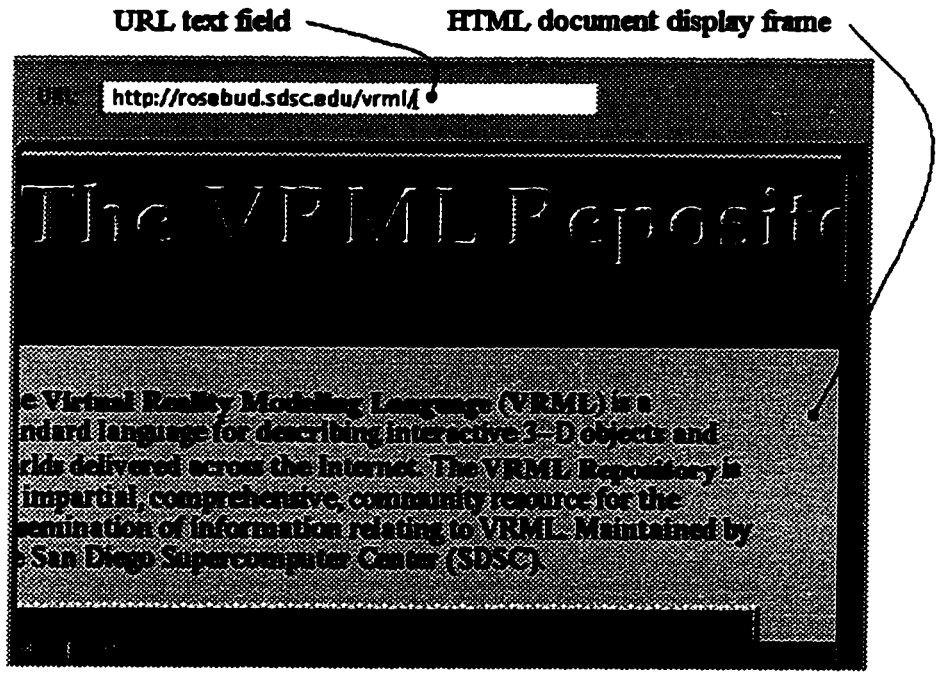


Figure 21. The URL applet

This application requires consistency because new users should be presented with the current HTML document in the frame. In this case, the Consistency class (called URLSimulator in the code) is simply a thread that keeps track of the latest URL requests by the last user. When a new user joins the session, this URL is sent to the URLFetcher

applet at the initialization stage through the initSetup() methods of the Client and the Server.

There is no messaging scheme for this particular applet since there is only one piece of information that is being sent: the URL of the target HTML document.

5.2.3 Whiteboard

This applet allows users to simultaneously draw on a board. No access control is performed on this applet since it is O.K. for more than one user to use the applet at the same time.

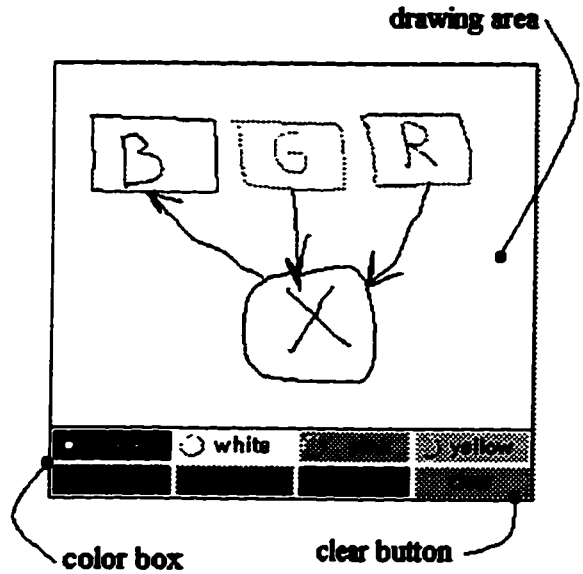


Figure 22. The whiteboard applet

The Consistency object for this applet is a thread that updates an Image object. The messaging scheme used for this applet is shown in the table below:

TABLE 5. MESSAGING SCHEME OF THE WHITEBOARD APPLET

Message	Meaning	Followed by
0	draw line	(x0, y0), (x, y) representing coordinates of the line, plus (r ,g, b), representing the color of the line
1	clear board	-

5.2.4 VRML viewer

This applet allows sharing of 3D objects in VRML format. This applet requires both access control and consistency. Access control is needed to ensure that only one user is interacting with the object at a time. Without access control there is significant risk of conflict between users' actions.

One feature of this applet is the ability to show different 3D objects as requested by users as opposed to just one 3D object. Users can choose one of the many 3D files presented in the applet's file choice menu and fetch that file (see figure below).

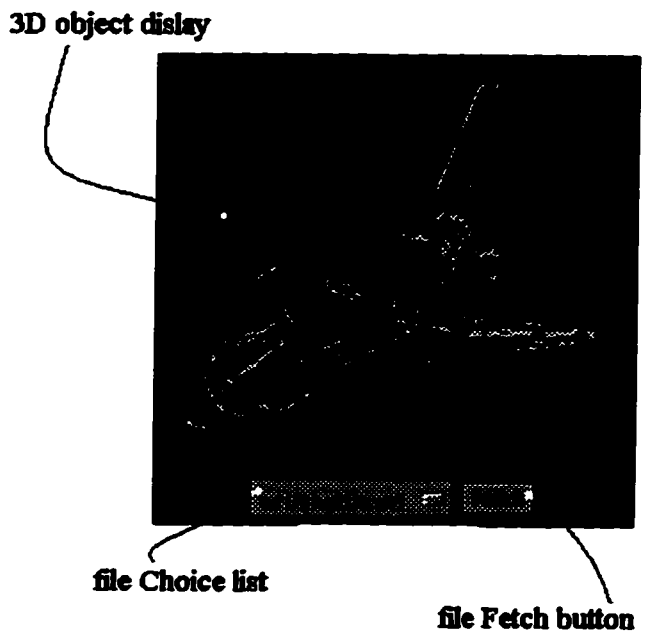


Figure 23. The VRML viewer applet

It should be noted that the 3D rendering and display of the applet was part of the WireFrame demo applet distributed by Sun Microsystems as part of the Java Development Kit 1.0.2 (JDK 1.0.2) [9]. That demo presents an applet that can display 3D files

represented in Wavefront (.obj) format. The code was significantly changed to add the following capabilities:

- double buffering to create smooth motion;
- complete navigation: move, rotate, zoom; as opposed to only rotate
- parsing of VRML (.wrl) files;
- displaying multiple 3D files (one at a time).

The applet is limited to wire-frame display of objects. However, the goal here was not to build a complete VRML browser as that is a very lengthy and complex procedure that is done by computer graphics specialists and is beyond the scope of this thesis. An already developed VRML browser that runs as a Java applet such as Liquid Reality[6] [14] can be used for that purpose.

The Consistency class for this applet (called VRMLSimulator in the code) contains the following data:

- a 3×4 geometric rotation matrix;
- a 2D coordinate pair;
- magnification factor;
- name of current VRML file in use.

At initialization time, the initSetup() method of the VRML server will fetch these information to the new client. While performing this initialization, the application is locked

[6] Liquid Rreality is a VRML 2.0 complient browser that runs as a Java applet inside Web browsers. It is developed by Dimension X corporation.

to make sure no changes are made during the initialization process. If a client has already locked the application, the initialization will be suspended until the lock is released.

The messaging scheme used for this applet is shown in table 5.

TABLE 6. MESSAGING SCHEME OF THE VRML APPLET

Message	Meaning	Followed by
0	interaction	(x, y), representing the present location of the mouse
1	change navigation mode	0=walk, 1=rotate, 2=zoom
2[1]	magnification factor	a float, representing the value
3	get new VRML file	a string containing the URL of the VRML file

1 only picked up by the Consistency object

It is interesting to note that there is the possibility to send certain messages to only the Consistency object. This is done by not handling that message in the run() method of the applet. Similarly, one could send messages that are ignored by the Consistency object, if ever needed. This is done by not handling that specific message in the run() method of the Consistency object.

It should be noted that the VrmlSimulator does not fetch a new VRML file when the applets do so. The reason lies in the implementation of the VRML viewer. When a new file is loaded into the viewer, the rotation matrix and the coordinates of the object don't change. That's why message 3 is not picked up by the Consistency object. What does change is the magnification factor which is sent to the VrmlSimulator by message 2 and is not picked up by other applets.

In addition to the regular Client applet parameters, the VRML applet takes a "file" and a "scale" parameter. The file parameter contains the names of the available 3D files, while the scale parameter indicates the initial magnification of the VRML object.

5.2.5 The Client Interface

The following picture is a sample screen shot of a typical tele-learning session incorporating all of the above sample applications:

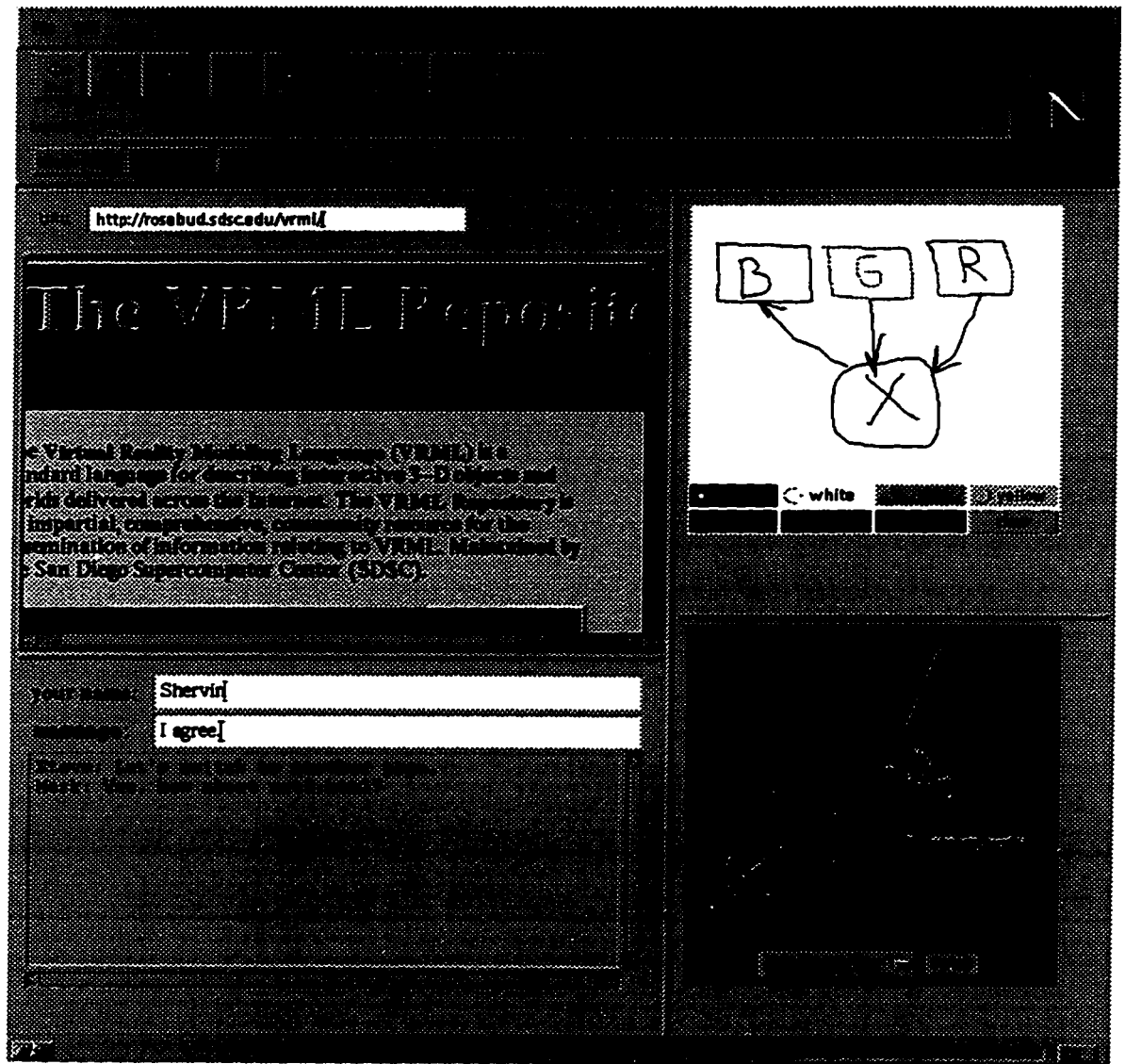


Figure 24. A sample tele-learning session running in the Netscape browser

As one can observe from the above picture, each applet is individually located in a separate browser frame. Although not all the Web browsers support frames, the ones that are Java-enabled do support frames; therefore, there is no risk of loosing potential users.

The main advantage of using frames is that new ones can be added and old ones can be modified or deleted with little effort, making it easy to introduce new applets. This facilitates the maintenance of the system. The frame hierarchy for the session in figure 23 is shown in the diagram below:

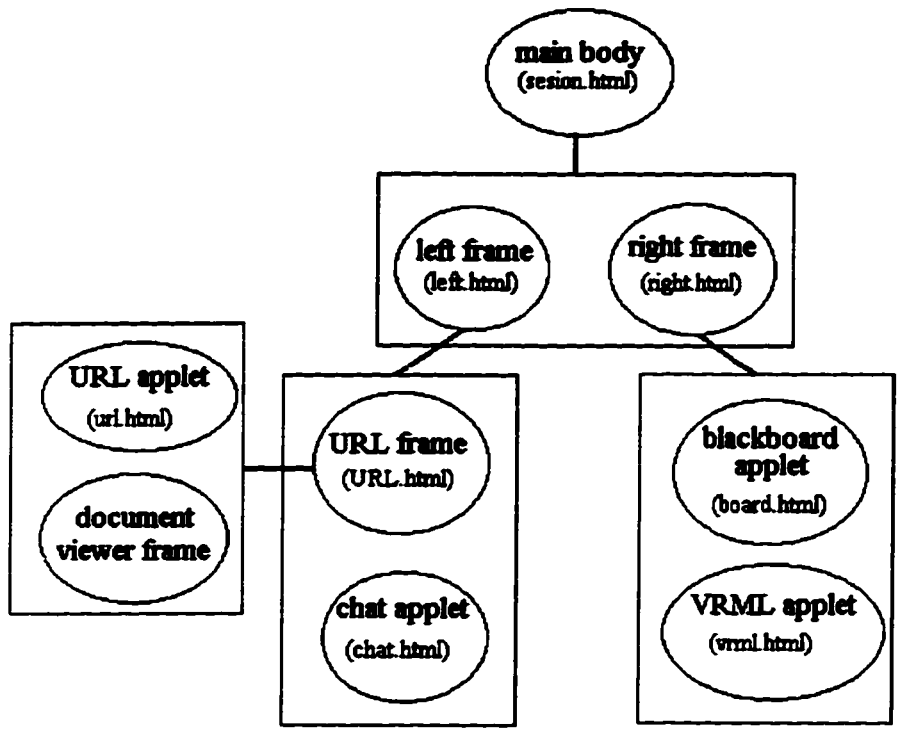


Figure 25. Frame hierarchy of the HTML documents

For the HTML code of the individual frames, as well as the applets' HTML documents please refer to appendix A.

Chapter 6. Performance Evaluation

The performance of the JETS system was tested using both subjective and objective methods. The tests were conducted in two areas: performance of the clients on different platforms, and performance of the servers over different networks, namely Internet and ATM.

6.1 Platform Issues

The performance of the system was tested on several different platforms. Parameters such as user interaction, screen-updating speed, and ease of use were tested in a subjective way. The server was launched on a SUN Sparc 20 machine with Solaris 2.3. The best results occurred for the most powerful machines, of course. The system was tested with Netscape Navigator 3.0 on the following machines on local Ethernet:

- SUN Ultra I with Solaris 2.5
- SUN Sparc 20 with Solaris 2.4
- Pentium 100 MHz PC with Windows 95; two browsers:
    - Netscape Navigator 3.0 by Netscape corporation
    - Internet Explorer 3.0 by Microsoft corporation
- Apple Power PC
- Apple 68000

The following discussion is based on user's point of view and not on objective performance tests:

The best performance was seen on the SUN Ultra machine. Using Netscape Navigator 3.0, a tele-learning session was launched. The performance of the applets was very good. Both

drawing on the whiteboard and interacting with the 3D objects were quite smooth. Sending messages and requesting HTML files were as real-time as it can be. Also, the graphical behavior of the browser and the screen-updating speed were very satisfactory.

The second best performance was seen on the Pentium 100 MHz PC with Netscape Navigator 3.0. Although the interaction with the 3D object was a little bit slower than the SUN Ultra case, the performance was quite similar to that of the SUN Ultra.

There was a problem with running the system with the Internet Explorer browser on the Pentium machine. Each of the applets ran in the Internet Explorer with no problem; however, when the system was run as a whole with all applets, always one of the applets was not working. After some investigation, it was discovered that there is a limit for the number of network connections opened by the Internet Explorer browser at the same time. Because of this limit, only four network connections can be made at a time. Interestingly enough, it was found out that Netscape corporation was accusing Microsoft corporation of having deliberately created this limit to prevent the execution of some of Netscape corporation's software!

The SUN Sparc 20 machine came in next, with considerably slower yet acceptable 3D object interaction. Also, the drawing and screen-updating operations were a bit slower than the above two cases. The chat and shared HTML applets performed very well. Overall, it gave an acceptable level of user interaction and performance.

Slow performance of the system was observed on the Power PC Macintosh. Although the test is subjective and differs from person to person, it is safe to say that the interaction with the 3D object was slow, just around user tolerance level. Drawing and screen-updating was not any better though the performance of chatting and HTML sharing

applets was acceptable. One "annoying" property of Netscape on the Power PC Macs was the fact that every time the user resizes the browser while the browser is downloading the applets, the browser reloads all the applets again, even if there were only one more class or file left to download. This creates a lot of delay, specially for a hasty user.

The worst performance was seen on 68000 family Apple computers. This of course was expected after observing the slow performance on the Power Macintoshes. In this case, the screen-updating speed as well as 3D object interaction was unacceptable; although the shared-HTML and the chatting applets gave an acceptable performance.

The lack of good performance on the Macs in general is mainly due to the implementation of the Java virtual machine for Apple computers. The current implementation is slow and not suitable for all real-time applets. Better implementations, including JITs for Mac, are in the development phase and should be expected in the near future.

6.2 Network Performance

The system was tested on both local and wide-area networks and for both ATM and regular Internet connections. ATM had a very good performance compared to the Internet for the non-local network case; it also performed a little bit better than local Internet for the local network case.

Besides the subjective testing of the JETS sample session, a test applet was designed to measure the effective *client-to-client delay (CCD)* of the system. This is defined as the average time it takes for a byte of data to reach a target client from an initiating client. Just a reminder that the data has to go through the JETS server to reach the target client. The test is performed as follows:

There is a *Sender* and a *Receiver* applet. The Sender applet sends a byte of data through the data channel, the Receiver acknowledges the receiving of the data by sending a byte of data through the data channel. The Sender, after receiving the acknowledgment, sends another byte of data, and so on. The procedure is run for a time *t*, which is adjustable by the user. If there are *n* bytes of data sent and acknowledged in time *t*, the CCD is:

$$CCD = \frac{t}{2n}$$

It should be noted that the CCD test result will not only be an indication of the transmission speed of the underlying network, but also an indication of the end-to-end delay of the entire system including delays caused by the underlying layers such as transport layer, network layer, ATM or LAN datalink layers, and so on. The Java code for the Sender and the Receiver applets is presented in appendix B. At the MCRLab, the JETS prototype was tested on both Ethernet and ATM backbones as shown below:

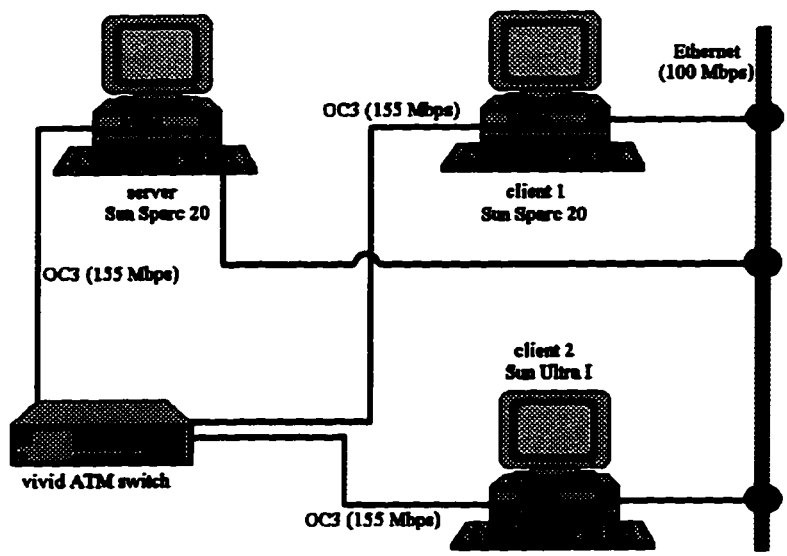


Figure 26. Local network configuration of the clients and the server

The server was run on a SUN Sparc 20 workstation with two clients, one running on a SUN Sparc 20 and the other on a SUN Ultra machine; both using the Netscape Navigator

3.0 browser. The Ethernet network was of type 100BaseT, capable of 100 Mbps transfer of data. The ATM transport was provided by a Vivid Workgroup ATM switch with OC-3 (155 Mbps) links.

Subjectively, no practical difference between ATM and Ethernet was observed and the system seemed to have the same performance on both networks. In order to determine the exact end-to-end delay, the CCD test was performed for three cases: local Ethernet, local ATM, and non-local ATM.

The network configuration for local ATM and local Ethernet was shown in figure 26. For the non-local case, OCRInet was used to establish a JETS session between the MCRLab and the COBRAnet laboratory at Nortel. Figure 27 shows the network layout of OCRInet.

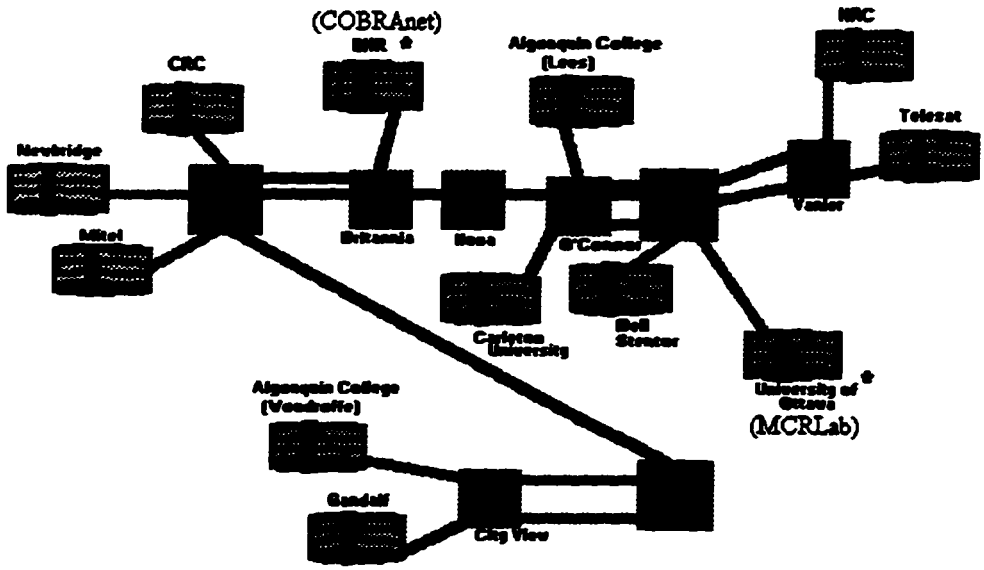


Figure 27. OCRInet Layout

First, the CCD test was performed for the two clients at the MCRLab on Ethernet. Then, the test was performed for the same clients over ATM. Finally, the same test was performed between a SUN Sparc 20 client at the COBRAnet laboratory and one of the

clients at the MCRLab, using the same server as for the previous tests. All tests were run for a 15 minute duration. The results are shown in the following table:

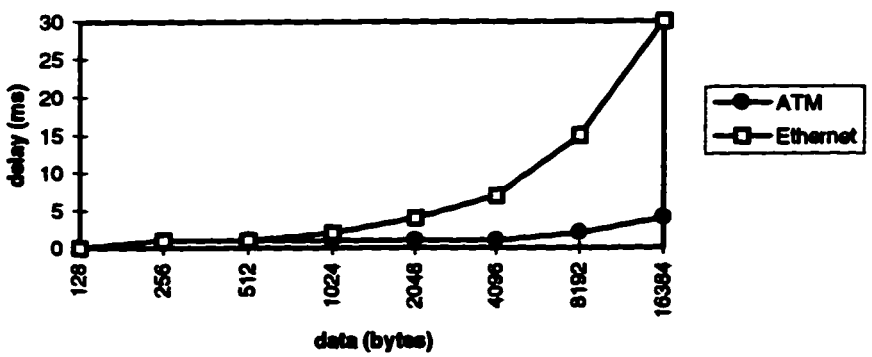
TABLE 7. CCD TEST RESULTS FOR DIFFERENT NETWORKS

Network	messages confirmed	duration (msec)	CCD (msec/byte)
Ethernet	8763	900044	51.33
local ATM	8850	900005	50.85
non-local ATM	8786	900001	51.22

Examining the above numbers, one can realize why subjectively no difference was felt between local ATM and local Ethernet. The end-to-end delays are very close; though ATM does have faster performance.

It should be noted that all of these results are for very small packet sizes. There is no question that ATM would perform far better than Ethernet for larger packet sizes. To illustrate this, IP's ping utility was used to test the end-to-end delay between the two clients on both Ethernet and ATM. The result is shown below:

Figure 28. Ping Time of Ethernet vs. ATM



The above plot shows that performance difference between the two local networks is not very significant for small packet sizes of approximately 0.5 Kbytes and smaller. At packet sizes of approximately 2 Kbytes or larger, the difference becomes significant.

6.3 Quality of Service

The ultimate endpoints of communication are the human perceptions and the brain's actual perception of sound, motion, and pictures. For humans, the end-to-end latency has an upper bound for acceptability. For interactive audio this limit is roughly 100 ms [18]. The following parameters were released by the Multimedia Communications Forum (MMCF) as a guideline for multimedia QoS[16]:

TABLE 8. SOME QOS PARAMETERS PUBLISHED BY MMCF.

	Class A (low quality)	Class B (medium quality)	Class C (high quality)
Audio/Video differential delay (msec)	N/A	between -400 and 200	between 100 and 150
DSD[7]/audio differential delay (msec)	less than 1000	less than 200	less than 100

Differential delay is the difference of arrival times between two media. The DSD/audio delay is the parameter of interest for this research. It states the maximum differential delay between audio and user interactions. For example, when a user rotates the 3D object and says "now I'm rotating the object", the difference between the time that his voice arrives and the time that the rotation message arrives at the other clients must be within the DSD/audio parameters.

For the JETS prototype, since the CCD is about 50 msec, the DSD/audio parameter is met for any kind of audio conferencing tool and for the highest quality (class C). The reason is that even if the audio conferencing tool, to be used in conjunction with JETS, would have a theoretical end-to-end delay of 0 msec, the differential delay would be 50-0=50 < 100. Hence, it can be concluded that the JETS prototype running on local networks, either

[7] DSD: Delay Sensitive Data, refers to data including pointers, control, and echoplex information.[16]

ATM or Ethernet, has a satisfactory performance and works well within the required parameters. This is also the case for ATM wide-area networks.

Due to security reasons at Nortel, no tests could be performed between the MCRLab and the COBRAnet laboratory on non-ATM network; i.e. the regular Internet. Nevertheless, it is obvious that regular Internet will perform drastically slower than ATM when it comes to wide-area networks.

Another interesting performance issue is the CPU utilization of the system. To measure that, the VRML applet was tested with one, two, and up to 5 clients. The CPU utilization of the server was always below 2%. So, it turns out that the number of open sockets on the server machine are more critical than the CPU utilization.

Chapter 7. Conclusion

The aim of this project was to theoretically design an practically implement a complete and functional telelearning environment that is accessible by users on the Internet with minimum amount of user-software. It is imperative to be seamlessly interoperable with currently existing and widely used technologies and standards to ensure the accessibility of the system to the broad education community. In the development process of the JETS system, this accessibility was achieved by a number of design decisions:

First, the VRML standard was deployed for the format of 3D objects. Though at its early stages, VRML has become the most popular 3D standard on the Internet. Using VRML further ensures the accessibility of the system. But VRML is more than just a 3D standard. It is envisioned as a virtual reality standard that will eventually address all the VR issues such as rendering, networking, perception, and so forth. Using VRML, one can create complete virtual environments that include 3D objects and other multimedia content. Hence, the decision of using VRML for the JETS system was not only made to ensure interoperability with existing standards, but also to facilitate the development of the next versions of the prototype, since advancements in VRML would translate to advancements in JETS.

Next, HTML was used as a standard for advanced documents. Perhaps this choice is the most obvious one. Practically all documents on the Web are written in HTML. By supporting HTML browsing, the huge number of already-developed HTML documents can be directly integrated and viewed with JETS. In addition, since the system uses the HTML browsing ability of the web browser, any future enhancements in HTML that is

supported by web browsers will automatically be supported by JETS. Like VRML, the development of HTML has also not reached its end yet and is constantly improving.

Then, the use of Java applets to provide application level services to the users ensures that any Internet user equipped with a Java-enabled Web browser has access to the system. Other than freeing the user from the responsibility of downloading and installing software and plug-ins, this approach utilizes a new concept in networked computers: let the application find the user and not the user find the application, as was the case traditionally. Furthermore, Java bytecodes can run on not only Web browsers and computers, but also simple network devices or mobile devices such as cellular phones, televisions, network terminals, and so on. This means that the system will still be interoperable with tomorrow's technologies without drastic modifications.

The other major advantage of using Java applets is the fact that they are platform-independent. This was in fact one of the properties that we were looking for to use for the telelearning system. Platform-independence may not be crucial for other types of collaborative environments; but for a telelearning system, it becomes essential simply because of the large number of users.

However, there is a price to pay for this platform-independence. Java applets are contained within a very restricted environment in order to prevent possible malicious actions performed against users. These restrictions limit the researcher in terms of the number of approaches that can be used to come up with a multi-user engine. For example, because of networking restrictions, a completely distributed system is not possible. There simply must be a centralized server and any communication between the users must be handled through this centralized server. Also, any kind of file manipulation on the client

side is prohibited; therefore, if an application absolutely must operate using files, it has to do it on the server.

These are some of the restrictions that enforce us to use a centralized server approach. Now, it must become clear what kind of service the clients need from the server. The two major demands of multi-user systems are **consistency**, which ensures all clients are presented with the same information, and **access control**, which restricts modification of documents by clients. Hence, there are four types of multi-user applications:

- need only consistency, like a shared whiteboard;
- need only access control, like modifying a database;
- need both, like a 3D viewer;
- need neither, like a chat session.

It is not only important that these services be provided by the server, but also that different applications be able to choose which service they need. It would be illogical to provide access control for an application that doesn't require access control. The problem with some of the existing client-server models is that the server provides for some sort of message passing, and the rest of the services must be handled by the application developer; i.e., the clients must take care of them. The JETS server supports both consistency and access control, so the clients can assume that the information they have is consistent. Also, using access control, a client can be sure that there is no conflict between its users actions with other clients' user actions.

The JETS server provides consistency by running a version of the client that does not display any graphics, receive or send any signals, or send any data. The advantage of this approach instead of running an exact copy of the client is saving resources and processing-

power. Since no one will ever see the consistency object which runs on the server, it would be wasteful to run a complete copy of the client for that purpose.

Access control is provided by means of a semaphore. Any client requiring to modify something that needs access control must request a lock on the semaphore. Depending on whether or not the semaphore is already locked, the client's request will be granted or denied. After performing the modifications, the client releases the semaphore.

Although very important for a multi-user environment, consistency and access control are not the only services that clients need. The server must also exchange messages among clients. The JETS server provides that service in a highly real-time fashion. There is one real-time thread running for each client, called the DataServer, that listens for messages from that client. When a message is received, the DataServer relays that message to all other clients. Since DataServers run independently, different clients can send messages at the exact same time. This is one of the major features of the JETS system.

The JETS system also supports signaling, which means clients can send control information to the server to request different services such as access to a semaphore. The signals are sent through a different channel that the data; in addition, there is a real-time thread running at the server for each client that handles these signals. Separating both the channel and the server threads responsible for data and signals makes possible the ability to send data and signal at the same time in real-time.

As one can observe, the JETS server is highly multi-threaded. In addition to the server, the client is also multi-threaded. There are at least two threads running that constitute the client: a main body and a receiver. The separation of the body of the thread from the task of receiving and handling incoming data enables the client to respond to its user's

interactions at the same time that it is updating the application from the messages received from other clients. Therefore, the client also behaves in a real-time manner.

Hence, it can be concluded that the JETS system is a highly multi-threaded system that provides services in real-time. This notion was tested by the implementation of a prototype.

Using the above architecture, generic Server and Client classes were developed. They were used to build specific sample applications: a whiteboard, a chat session, a 3D object viewer, and an HTML viewer. The system was tested on local area network, both Ethernet and ATM, and wide area ATM network. The test results indicated that the system is compliant with the MMCF requirements for the highest quality of service.

The prototype developed helped demonstrate that the system successfully works as expected. It also showed that the design theory and architecture behind the system is valid and implementable.

There are several improvements that can be made to further increase the quality of JETS. One improvement is to implement the Server class using C++ language. Being an interpreted language, Java is slower that C++ in nature. Hence, if it is not running on JAVAOS, which is an operating system devoted to run Java bytecodes, it will be slower than its corresponding C++ implementation. In order to be highly scalable, it is important that the server be as fast as possible since it might be serving a huge number of clients. Note that the implementation of the server in C++ does not take away the platform-independence feature of JETS as the clients will still be Java applets.

Another improvement would be the introduction of Remote Method Invocation. Using RMI, the clients can invoke methods of other clients directly. This will eliminate the

developers need to come up with a messaging scheme. However, RMI will also decrease the system performance as it introduces some overhead in the communication process. Currently, RMI is not part of the core Java package. Consequently, using it at this stage would drastically reduce the accessibility of the system as only people who do have RMI would use the system. Furthermore, RMI does not work with Java applets because of security reasons; although plans are underway to make RMI work with applets. Once RMI becomes available as part of the standard Java package and interoperable with applets, it would be a good idea to evaluate the use of RMI in JETS.

In short, the JETS system is a successful experiment of designing an implementing a high performance multi-user system. Being the first platform-independent real-time telelearning system, it is at its early stages of development and can be used as an example for future collaborative environments operational through the World Wide Web.

References

[1] Ari Luotonen, Kevin Altis, "World-Wide Web Proxies", "http://ezinfo.ethz.ch/general_info/www94/1st_day_proxies.html" , April 994.

[2] Arman Danesh, "JavaScript", Sams.net Publishing, ISBN: 1-57521-073-8, 1996

[3] "Asynchronous Transfer Mode (ATM)", ATM Forum, "http://ww.atmforum.com"

[4] "Atlantis Cyberspace VR Equipment", Atlantis Cyberspace, "http://vr-atlantis.com"

[5] David Flanagan, "Java in a Nutshell", O'Reilly & Associates Inc., California, ISBN: 1-56592-183-6, February 1996

[6] G.Blakowski and W. Steinbeck, "Harmonization of an Infrastructure for Flexible Distance Learning in Europe with CTA", Proc. 2nd Intern. Workshop on Advanced Teleservices and High-Speed Communication Architectures, Heidelberg, Germany, Sept. 1994.

[7] H. Ohzu and K. Habara, "Behind the Scenes of Virtual Reality: Vision and Motion", Proc. IEEE, Vol. 84, No. 5, May 1996.

[8] "IBM Network Station", IBM corporation, "http://www.internet.ibm.com/computers/networkstation/"

[9] "Java Development Kit 1.0.2 API", JavaSoft, "http://java.sun.com/products/JDK/CurrentRelease/api/"

[10] "Java RMI vs. CORBA" , JavaSoft Web site, "http://chatsubo.javasoft.com/current/faq.html#CORBA"

[11] "JavaScript", Netscape corporation, "http://home.netscape.com/eng/mozilla/3.0/handbook/javascript/"

[12] "Java support in the Internet Explorer", Microsoft corporation, " http://www.microsoft.com/ie/ie3/java.htm"

[13] Laura Lemay & Charles L. Perkins, "Java", Sams.net Publishing, ISBN: 1-5721-030-4, 1996

[14] "Liquid Reality", Dimension X incorporated, "http://www.dimensionx.com/products/lr/"

[15] Mark Pesce, "VRML: Browsing and Building Cyberspace", New Riders Publishing, 1995

[16] Multimedia Communication Forum Inc., "Multimedia Communication Quality of Service", MMCF document, September 24, 1995.

[17] OCRInet Web site : "http://www.OCRInet.ca/ocnthome.html"

[18] Olof Hagsand, "Interactive Multi-user VEs in the DIVE System", IEEE Multimedia, pp. 30-39, Spring 1996

[19] "QuickTime VR", Apple Computer, "http://quicktimevr.apple.com/"

[20] "RealVR Traveler", RealSpace Incorporated, "http://www.rlspace.com/"

[21] "VRML 1.0 API", The VRML Architecture Group, "http://vrml.wired.com/vrml.tech/vrml110-3.html".

[22] "The VRML Repository", San Diego Super Computing Center, "http://rosebud.sdsc.edu/vrml/"

[23] "VRML 2.0: The Moving Worlds Specifications", Silicon Graphics, "http://vrml.sgi.com/moving-worlds/spec/"

[24] "Web TV", Web TV Networks (Sony/Philips) , "http://www.webtv.net/"

[25] "What Is Java", Sun Microsystems, "http://java.sun.com/aboutJava/index.html"

[26] W. Steinbeck, "ECOLE: Applying Multimedia at the Point of Learning-A distributed Multimedia Environment for Flexible Distance Learning", IBM European Networking Center, TR 94.xxyy, 1994.

Appendix A: HTML code for the browser interface

session.html

```
<HTML>
        <HEAD>
        </HEAD>
        <frameset cols=60%,40%>
                <frame src="left.html">
                <frame src="right.html">
        </FRAMESET>
</HTML>
```

left.html

```
<HTML>
        <HEAD>
    </HEAD>
        <frameset rows=55%,45%>
                <frame src="url/URL.html">
                <frame src="chat/chat.html">
        </FRAMESET>
</HTML>
```

right.html

```
<HTML>
        <HEAD>
    </HEAD>
        <frameset rows=50%,50%>
                <frame src="paint/paint.html">
        <frame src="vrml/vrml.html">
        </FRAMESET>
</HTML>
```

URL.html

```
<HTML>
        <HEAD>
    </HEAD>
        <frameset rows=12%,88%>
        <frame src="url.html">
                <frame name="WIN">
        </FRAMESET>
</HTML>
```

url.html

```
<title>URL Fetcher</title>
<applet code=URLApp.class width=350 height=20>
<param name=winName value="WIN">
<param name=consistency value="true">
<param name=port value="6792">
</applet>
```

chat.html

```
<title>Chat </title>
<applet code=Chat.class width=500 height=275>
<param name="port" value="6790">
<param name="host" value="mango.genie.uottawa.ca">
</applet>
```

board.html

```
<title>Paint </title>
<applet code=NetPaint.class width=300 height=275>
<param name="port" value="6789">
<param name="host" value="mango.genie.uottawa.ca">
<param name="consistency" value="false">
</applet>
```

vrml.html

```
<title>VRML Browser</title>
<applet code=VrmlApp.class width=300 height=300>
<param name="files" value="wrl/camaro.wrl wrl/fish.wrl wrl/human.wrl wrl/penguin.wrl wrl/rat.wrl
wrl/shuttle.wrl wrl/snail.wrl wrl/torus.wrl" >
<param name="scale" value="1.0">
<param name="consistency" value="true">
<param name="port" value="6791">
<param name="name" value="VRML">
<param name="signal" value="true">
<param name="host" value="mango.genie.uottawa.ca">
</applet>
```

Appendix B:

Java and HTML Code of the CCD Test Applet

B1. Java Code

Sender

```
import java.awt.*;
import java.io.*;
import java.util.*;

public class Sender extends Client{

        TextField inputfield;
        TextArea outputarea;
        long time;
        Date date;
        boolean go=false;

        public void init() {
                super.init();
                GridBagLayout gb=new GridBagLayout();
                GridBagConstraints gc=new GridBagConstraints();
                setLayout(gb);
                Label l1=new Label("Time (msec):");
                gc.fill=GridBagConstraints.NONE;
                gc.gridwidth=GridBagConstraints.RELATIVE;
                gb.setConstraints(l1, gc);
                add(l1);

                inputfield=new TextField();
                gc.fill=GridBagConstraints.HORIZONTAL;
                gc.gridwidth=GridBagConstraints.REMAINDER;
                gb.setConstraints(inputfield, gc);
                add(inputfield);

                outputarea=new TextArea();
                gc.fill=GridBagConstraints.BOTH;
                gc.gridwidth=GridBagConstraints.REMAINDER;
                gc.gridheight=GridBagConstraints.REMAINDER;
                gb.setConstraints(outputarea, gc);
                outputarea.setEditable(false);
                add(outputarea);
        }

        public boolean initSetup() {
                return true;
```

```
	}

	public boolean action(Event e, Object what) {
		if (e.target==inputfield) {
			time=Long.parseLong( inputfield.getText() );
			outputarea.appendText("Testing for "+time+" msec\n");
			go=true;
			inputfield.setText("");
			return true;
		}
		else return false;
	}

	public void run() {
		String line;
		try {
			for(;;) {
				if ( go ) {
					int n=0;
					Date d=new Date();
					long t1=d.getTime();
					while ( (d.getTime()-t1)<time ) {
						out.writeByte(1);
						while ( in.readByte()!=1 );
						n++;
						d=new Date();
					}
					long t2=d.getTime();
					outputarea.appendText(n+" bytes exchanged in "+(t2-t1)+"
msecs\n");
					double ccd=( t2-t1)/(2*n);
					outputarea.appendText("CCD: "+ccd+"\n");
					out.writeByte(2);
					out.writeDouble(ccd);
					go=false;
				}
				else
				 try {Thread.sleep(100);} catch (InterruptedException e) {};
			}
		}
		catch (IOException e) {outputarea.setText(e.toString());}
	}
}
```

Reciever

```
import java.awt.*;
import java.io.*;

public class Reciever extends Client{
```

```
        TextField inputfield;
        TextArea outputarea;

        public void init() {
                super.init();
                GridBagLayout gb=new GridBagLayout();
                GridBagConstraints gc=new GridBagConstraints();
                setLayout(gb);

                Label l1=new Label("Ready");
                gc.fill=GridBagConstraints.NONE;
                gb.setConstraints(l1, gc);
                add(l1);

                inputfield=new TextField("                                    ");
                gc.fill=GridBagConstraints.HORIZONTAL;
                gc.gridwidth=GridBagConstraints.REMAINDER;
                gb.setConstraints(inputfield, gc);
                add(inputfield);
        }

        public boolean initSetup() {
                return true;
        }

        public void run() {
                String line;
                byte x=0;
                try {
                        for(;;) {
                                while ( ((x=in.readByte())!=1) && (x!=2) ) {
                                } //end while
                                if (x==1) out.writeByte(1);
                                else if (x==2) {
                                        double ccd=in.readDouble();
                                        inputfield.setText("CCDt: "+ccd+" msec");
                                }
                        }
                }
                catch (IOException e) {outputarea.setText(e.toString());}
        }
}
```

TestServer

```
import java.util.*;
import java.io.*;
import java.net.*;

public class TestServer extends Server {

        public TestServer(String name, int port, int numberofClients) {
```

```
            super(name, port, numberofClients, false, false);
            start();
        }

        public void initSetup(int i) {
             int x=1;
        }

        public static void main(String[] args) {
            boolean complete=true;
            int PORT=0, NUMOFCLIENTS=0;
            if (args.length!=2) complete=false;
            else {
                try {
                    PORT=Integer.parseInt(args[0]);
                    NUMOFCLIENTS=Integer.parseInt(args[1]);
                }
                catch (NumberFormatException e) {complete=false;}
            }
            if (complete) new TestServer("Test", PORT, NUMOFCLIENTS);
            else System.out.println("Usage: java Server name port-number number-of-clients");
        }
}
```

B2. HTML Code

sender.html

```
<title>Sender </title>
<applet code=Sender.class width=500 height=275>
<param name="port" value="6795">
</applet>
```

receiver.html

```
<title>Reciever</title>
<applet code=Receiver.class width=400 height=175>
<param name="port" value="6795">
</applet>
```