A 3-D PLOTTING TOOL FOR INTERNET BASED ON CLIENT-SERVER COMPUTER MODEL

SHIHUA WU

A MAJOR REPORT

IN

THE DEPARTMENT

OF

COMPUTER SCIENCE AND SOFYWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE AT

CONCORDIA UNIVERSITY

MONTREAL, QUEBEC, CANADA

SUMMER 2005

Your file Votre référence
ISBN: 0-494-10303-5
Our file Notre référence
ISBN: 0-494-10303-5

ABSTRACT

A 3-D Plotting Tool for Internet Based On Client-Server Computing Model

Shi Hua Wu

We describe the specification, design, deployment, and implementation of a three-dimensional plotting utility based on a client-server distributed computing model. This plotter provides a World Wide Web environment enabling the user to get 3-D colored curves from mathematical expressions. The utility provides an intuitive look-and-feel user interface. Users do not have to acquire extra knowledge to use it. The utility is reliable, portable and extensible.

In this report, we discuss the project motivation. We compare three popular Java middleware technologies to show why we chose Java Remote Method Invocation (RMI) technology. Java Native Interface (JNI) technology and Java 3 Dimension Application Interface (3-D API) is also described. We discuss the advantage of the reuse component model, and demonstrate how to reuse an object-oriented model (C++ parser model) in a Java program.

Finally, we provide a short conclusion and discuss the future perspective and improvement of this utility.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Professor Peter Grogono, for his encouragement, technical advice, patience and constant support throughout this work. The author was deeply benefited from his knowledge about graphic system.

I also like to express my thanks to the following individuals:

- Ms. Halina Monkiewicz, the Graduate Program Secretary, for her kind support.
- Mr. Meng Cai, my former colleague, for his useful advice.

And, I am deeply indebted to my family: my husband and two kids, for their continuous support and encouragement.

Table of Contents

List of Figures

1 PART I. INTRODUCTION

1.1 Introduction

In this project, we describe the development of an on-line 3-D plotter with a friendly user interface with which the user can input formulas and obtain corresponding three-dimensional graphic surfaces. The precision of the optimal animation can be changed by the user. The software used to develop the client component is the Java 3-D API package. The user interface is based on the Java AWT package. Java Remote Method Invocation (RMI) technology provides the communication between client and server components.

In first part of this report, we discuss the motivation for developing a 3-D plotter based on client-server model, the reason for choosing the programming language and the middleware technology. The whole system architecture and the functional requirement are also be specified in this part.

1.2 Background

The widespread development of the Internet has placed web sites amongst the largest mass media of the world. With the explosive growth of information sources available, it has become increasingly necessary for users to utilize on-line tools. In this situation, a 3D plotting utility based on client-server model has become necessary.

A plotter is used to help people to visualize a mathematical function or compare a group of functions. Compared to customized plotters, like Mathematica [1], MatLab [2], and

Maple [3], the web-based plotter is reliable and easy to use. Meng Cai developed a 2-D plotting utility based on client-server model [4], making it feasible for people to share the single copy of the application from World Wide Web. The server component can be installed on a host computer, and a regular user just needs a conventional browser to use it. And, whenever the application is updated on the server computer, the user can automatically load the latest version of the client component and use it.

But the graphics character of the two-dimensional plotter is not enough to satisfy modern users, who may have seen three-dimensional plotters written, for example, in OpenGL. Although these are not on-line plotters, they are colorful, beautiful, and stereoscopic. According to this need, our goal was to develop an on-line 3-D plotter.

1.3 A Summary of the previous work

Meng Cai's work [4] was based on the work of former master student Ahn Phong Tran [5]. His report describes a standalone 2-D Graph Plotter application which can be used to plot simple algebraic expression. The software consists of two major components: a parser/evaluator and a user interface. The parser can parse an expression, and the evaluator can set the variable values in the expression and calculate the value of the expression. The user interface for the plotter uses Microsoft Foundation Class.

The parser uses the following the following grammar to parse an expression:

Expr -> ['-'] Term {('+'|'-') Term}.

Term -> Factor {(''|'/')Factor}.*

Factor -> Primary ['^'Primary].

Primary -> NUM |VAR ['('Expr ')']|'('Expr')'.

Where NUM represents a number, VAR represents either a simple variable or a built in function (fg. sin(), cos(), log(), sqrt()...). The form [X] means that the expression X is optional — it may appear or it may not. The form (X1 | X2) indicates an alternative — either X1 or X2 must appear. The form {X} stands for "Kleene closure" — X may appear zero or more times.

The parser/evaluator was built using the recursive descent method. It performs the following tasks:

- Given an expression, determine if it is syntactically correct, and construct a parse tree for it.
- Given the values of the variables, parameters and constants in the expression, find the value of the expression.

Tran's code was written in C++ [5]. Meng Cai extended the above 2-D graph plotting tool to a component-oriented application [4]. He developed a client component which can interact with a user, and a server component that is responsible for parsing and evaluating expressions. The client and server components are relatively independent. The client can communicate with the server via a well-defined interface. The user can load the client component with a web browser. The client component running on the user's computer has a user interface to keep some state information and can to let the user input

expression and view the corresponding curves. The server component running on the host computer is called by the client and is responsible for parsing and evaluating expressions.

From Meng Cai's work [4], what we can reuse is the parser DLL. Meng Cai uses Java Native Interface (JNI) facility to build a java interface from Tran's non-Java legacy system[5]. JNI is a native programming interface. It is organized like a C++ virtual function table, through which Java code running inside a Java virtual machine can interact with applications and libraries written in other programming languages, such as C, C++ and assembly language. Meng Cai uses JNI technology to integrated C++ parser module to a dynamic link library (DLL). The old parser module including *parseval.h*, *parsInterfImp.h*, *parseval.cpp*, *parsInterfImp.cpp* is integrated to paser.dll.

1.4 Objective of this project

The objective of this project is to build a 3-D plotting tool based on the client-server computing model. The plotter will be able to parse and validate an expression input by a user, evaluate the expression, and display the resulting surface or surfaces within a Web browser.

Valid expressions contain two independent variables. They may also contain one additional identifier used as a parameter, and maximum of three identifiers as constants. Users should be able to specify the range of the independent variables by entering minimum and maximum values. In addition, users should also be able to specify the resolution (the intervals) of the graph.

The tool will also be client-server enabled and based on the component computing model. It provides user-interface to allow user to enter expressions, to select precision, and to view the resulting surfaces.

This application will be implemented in Java. The communication between client and server components will be based on standard Java Remote Method Invocation (RMI) technology. The 3D display will use the Java 3D APIs.

The goal of this project is to build a convenient 3-D plotting tool that is efficient, easy to learn and use, reliable, and extensible. It may also serve as a prototype system to demonstrate how the latest technologies can be integrated effectively to develop sophisticate application in client-server circumstance.

1.5 Functional Requirement

The application is intended for plotting three dimensional graphics. The following functional requirements are specified for the system:

- The plotter must be able to parse and validate a formula input by the user. An expression consists of identifiers (with one or more letters), constant numbers, function names (support is provided only for a fixed set of built-in functions), parameter k (used for animation) and operators. For example:

  x*exp(y)

  Sin(x+y)-coos(x-y)

  a*x*x+b*y+c

k*sin(x^2+y^2)

- If the formula that the user inputs has syntax errors, an error message corresponding to the problem will appear in the error box.

- The user can define an independent variable, and must specify the range of the independent variable by entering a minimum and maximum value. If an identifier is not defined by the user, by default; it is assumed to be a constant and has the value zero.

- The user can also define three different constants and set values for them. By default, the value of each constant is zero.

- After the draw function is activated, if the expression is valid and the identifiers are defined, the plotter parses the formula and displays the three-dimensional curves or surfaces to the user.

- The three-dimension graphs have the lighting property. More than one light direction can be put on the light objects. Different surfaces may have different colors. The color of the graphic and the background can also be defined in the programming stage.

- The user can use the mouse to rotate the graphic about any axis. Some keys on the keyboard can also be used to rotate or zoom in/zoom out the graphic. If the graphic is show out of the range, one key provides the function to reset the original state.

- The application also provides animation functionality. One parameter can be identified by the user. The user specifies the range of the parameter by entering a

minimum and maximum value. The precision of the animation can be controlled by the user by setting the number of steps.

- To make it easier for Internet users, the whole interface is embedded in an HTML file and can be accessed by all internet browsers.

1.6 Nonfunctional Requirement

1.6.1 Usability requirement

This section describes the system usability requirements. These requirements usually relate to the general aspects of the interface between the user and the system.

1.6.1.1 Client browser should support html page.

1.6.1.2 Client browser should support Java Applet.

1.6.2 Performance requirement

This section describes the system performance requirements. These requirements usually relate to the execution speed and capacity of the individual component of the system.

1.6.2.1 Web server performance

1.6.2.1.1 The web server in this system should be able to handle at least 5 simultaneous user sections. This means if five users access the server from different computer, the server should not crash.

1.6.2.1.2 The system should require no more than 10 seconds to retrieve and respond to a client request for a static web page.

1.6.2.1.3 The system should require no more than 15 seconds to respond to a dynamic page.

1.6.3 Hardware requirement

This section describes the system's hardware requirement. These requirements often state the minimal hardware required to implement the system.

1.6.3.1 Server side hardware requirement

1.6.3.1.1 The minimal server configuration: Pentium I 200 MHz, 64 MB RAM, 800X600X256, 1GB HD, otherwise the server maybe does not work.

1.6.3.2 Client side hardware requirement

1.6.3.2.1 Any computer can run explore 4.0 (or higher), or Netscape 4.0 (or higher) browsers.

1.6.3.2.2 Minimum modem connection rate: 28KBs

2 PART II. SOFTWARE DESIGN

2.1 Design Rationale

Our goal in designing this utility tool is to achieve a good reliable utility for easy modification and expansion by using component-oriented designed architecture. As the functional requirements document specified, this plotting utility should be user friendly, rapidly developed, and have good portability. It should be efficient and it should provide a good appearance to the viewer.

2.1.1 OpenGL

"OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry's most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms. OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions. Developers can leverage the power of OpenGL across all popular desktop and workstation platforms, ensuring wide application deployment." [6]

OpenGL is a widely used and cross platform graphics programming API. It is a bunch of function calls which can create and controll the coloring of pixels on the screen. OpenGL has some advantages towards other graphics APIs:

The biggest advantage is that OpenGL is platform independent. OpenGL is available on each platform because OpenGL does not contain dependent functions like for example creating a window. So if you plan to write an OpenGL application just include GL/gl.h at the top. You just need to download GLUT (OpenGL Utility Toolkit) and GLX, GLX is the OpenGL extension to the X window system (which merges xlib programming with OpenGL programming). Or you can have the GLUT API. The GLUT API provides the ability to create a window which can show OpenGL content and handle basic events.

Another advantage of OpenGL is that it is stable. Most videocards in the world have native OpenGL support, it is an industry standard, the ARB (architectural review board) controls the evolution of OpenGL.

Even for Linux system there is also a free implementation of the OpenGL API available. That is the MESA3D package. [7]. It isn't an official OpenGL version but it is compatibility with the official OpenGL implementation

2.1.2 Java 3D API

Java 3D API is the fourth generation of application program interfaces. [8] As show in

API	Hardware	Language	Connections	3D Graphics	New users
Siggraph CORE	Raster Terminals Minicomputers	FORTRAN	Serial I/O	3D Primitives Surfaces	MCAD Companies
PHIGS/ PEX	Workstations	C	I/O Busses/ Ethernet	Display Lists Solids Basic 3D Acceleration	MCAE Workstation Users
OPEN GL	Advanced 3D Accelerated Workstations	C++	Integrated 3D Graphics Systems	Immediate Mode Texture Mapping Advanced Lighting Advanced 3D Hardware	Animation Simulation
Java 3D	Java Terminals Network Computers Set-top Boxes Game Consoles	Java	Internet Networks	Scene Graphs Geometry Compression Behaviors Spatial Sound	Game Developers Web Page Designers Information Distributors

Fig2.1: Four generation API

the Figure 2.1, compared to the first generation of API Siggraph CORE, the second generation of API PHIGX/PEX, and the third generation of API OpenGL, it has five basic requirements that exist for this new standard. These includes a new hardware architecture, a new programming language, a new graphics connection to the host computer, new ideas and technologies concerning 3D interactive graphics, and new types of users who are not experts in the previous generation of graphics APIs.

Java 3D has taken a different approach to defining dynamic behaviors. By allowing a user to program a Java applet with the behavior defined in the applet, a lot of behavior

options can be created. This can be used to provide special rendering, sound effects, interaction, and other creative options. These behaviors can be shared between scene graphs. The combination of scene graphs and behaviors can be defined as a Java object, and this object can be linked with other objects to create capabilities that can be shared by developers. So all the scene graph segments, behaviors, and viewing models can be reuse by the developers.

Java 3D was designed to allow a developer to create dynamic 3D interactions with little knowledge of the low-level rendering pipeline. This frees developers to create their content and the user interactions without being forced to manage the details of the rendering process.

This kind of API is straightforward, and there are numerous examples and sample code available. Enhancing a web site with a 3D interface and allowing a user to completely visualize a product are some of the possible ways to make a web site more appealing. The developer writing 3D graphics applications will appreciate the "write once, view anywhere" aspects of Java 3D.Concidering the advantages of java 3D API, we decide to choose it to develop our user interface.

Actually Java 3D API is based on OpenGL. Since it is a high-level API, it is easier to use it to develop 3D graphics programs. Using a high-level language, a programmer can be more productive when given a diverse set of powerful tools and API's. It also simplifies the programmers' job since they are able to work with higher level constructs like Object Oriented constructs instead of low-level constructs.

We choose Java 3D also because Java user interfaces are more mature. It provides properties such as lighting, texture, and animation that permit graphic products to match users' needs more closely. Since Java is also supported by many distributed object technologies, it is suitable for almost all kind of computers even including desktop computers and laptops, which will become more and more popular in the future. [9] There are many advantages for Java program:

- Portability – Java runs on a wide variety of computers and computing devices. Java Virtual Machines (JVM) can run Java applications that were written on other platforms. The same program that runs on one device's JVM will run on another device's JVM.

- Object oriented – Java provide a cleaner approach to object-oriented programming, Java 3D API provides a high level object-oriented interface so that programmers do not have to concentrate on the low-level details. A "layer of abstraction" or "separation from the details" enables programmers to think more in terms of the problem domain. This in turn makes them more productive, since they don't have to understand how the problem is actually solved by the underlying software and hardware. OO approch increases the portability of the program.

- Easy to use – Java programming language has fewer memory management responsibilities (no pointers), a less confusing syntax, and simpler method resolution rules than other languages, etc. It has proven to be easier to use than other more complex languages, and that helps developers become more productive It has many

high level features that make it easier to write programs in Java than with languages using lower level API's. A Java 3D graphics programmer does not have to write the low-level code that manipulates and draws lines, triangles, rectangles, and other shapes from the virtual universe to the display device. The public (or standard) and private (or proprietary) Java 3D archives contain a great number of useful classes and methods to generate 3D objects, manage them in a 3D world, and view them. Java 3D separates the details of how objects are rendered from the objects themselves. This means that the programmer can create virtual entities in a virtual universe, and have them rendered without specifically telling the computer how the rendering process is to be performed.

- Security – Java applications and applets run in an environment that is easy to control in order to prevent accidental or malicious abuse of the system and its resource.

- Network oriented – Java was originally designed to run on interactive appliances, connected together with other devices in a distributed environment. It is often used to develop stand-alone applications and Web based applets that will run on a wide variety of machines, and it is a great way to make an interactive application for a Web page. When an application or web page wants to have 3D graphics and 3D interactivity, then the Java 3D API is naturally the way to add those functions. It integrates perfectly well with both web browsers and web servers and provides excellent support for the development of web-based application.

We can see that the Java 3D API will become the interactive 3D graphics API of the web and the desktop, since it can fulfill the promises of Java to the developer, and meet the performance demands of the user.

Java 3D API also has optimizing features that strike a balance between ease-of-use for the programmer, portability, and efficiency on the target machine. Java is well-suited for writing those portable applications that will be run on a variety of platforms and those Java Applets for the web pages. Therefore, these types of programs that also have a user interface that requires 3D graphics, or that would be significantly enhanced by the use of 3D graphics, are well suited to being written with Java and the Java 3D API.

The strengths of the Java 3D API make it an effective choice for programming 3D graphics. There are Java 3D implementations for SUN Solaris, Windows, SGI IRIX, HP-UX, and Linux. The "layers of abstraction" means the display devices are not limited to just computer monitors. Stereoscopic display devices, projection 'rooms', and multiple screens can also be used as easily as a single computer screen to view the output of a program.

Of course Java also has its disadvantages: Since programs written in Java language runs on a virtual machine, it runs somewhat slowly compared to other programs. However, computer today has faster processors, more memory, and graphics hardware accelerators Thia concern become not so annoyingly.

Another problem is that the programs do not always work correctly even if they are written correctly. It is difficult to write a perfect program, especially if it is as big as a

virtual machine. Since a Java virtual machine may be written incorrectly, so that programs written in Java langauge tend to suffer from slightly different problems depending on the platforms on which the Java virtual machines run.

For our utility tool, it is not so big and don't have very complicated functionate to comsume a computer source, we choose Java 3D to implement client subsystem.

2.1.3 Remote method invocation — the middleware choice

Java is supported by many distributed object technologies. By using Java, we will have more alternative technologies to choose from. In the Java platform's distributed object model, a remote object is one whose methods can be invoked from another Java virtual machine, potentially on a different host. An object of this type is described by one or more remote interfaces, which interfaces are written in the Java programming language that declare the methods of the remote object. [10]

Remote method invocation (RMI) is the action of invoking a method of a remote interface on a remote object. Most importantly, a method invocation on a remote object has the same syntax as a method invocation on a local object. RMI is a kind of middleware well used in distributed application.

Middleware is a kind of software that provides a framework to shield programmers from lower level details of networking protocols, as well as heterogeneous data representations, hardware, and software environments across network. In order to figure out why we choose RMI, we compare CORBA, Sun's Java/RMI and Microsoft's

COM/DCOM – three most popular distributed object paradigms for application development today.

CORBA

CORBA, is specified by Object Management Group (OMG), which is the largest consortium in the software industry. [11] CORBA is based on the Request-Response architecture. There is an object implementation on the server, which the client requests to execute. The client and the server object implementation do not have any restrictions on the address space, for example the client and the server can exist in the same address space, or can be located in separate address spaces on the same node, or can be located on separate nodes altogether.

CORBA objects are objects that supports the CORBA:: Object IDL (Interface Definition Language) interface and the Remote references are called Object References. There is a specification of the CORBA IDL Language and how it is mapped with other languages. It provides a means for the interface definitions. The Proxy or a local representative for the client side is called the IDL stub; the server-side proxy is the IDL skeleton. The proxy represents an object created on the client side, which is used for more functionality like support for Dynamic invocation.

For marshalling the request and the response, the information is delivered in a canonical format defined by the IIOP protocol used for CORBA interoperability on the Internet. IDL stub makes use of dynamic invocation interface for marshalling on the client side.

Similarly on the server side, IDL Skeletons use the Dynamic Skeleton Interface for unmarshalling the information. The request and the response can also contain Object Reference as parameters; remote object can be passed by reference. In order to make requests and receive response from other objects remotely to locally, the objects used Object request broker (ORB), which is the object bus.

CORBA ORB provides [12]:

- High-level language bindings
- Static and dynamic method invocations
- Local/ remote transparency
- Self-describing system
- Built-in security and transactions
- Polymorphic property
- Coexistence with existing systems

Consequently, CORBA can provide many benefits:

- Programming-language independent interface.
- Legacy integration
- Rich distributed object infrastructure
- Location transparency.
- Network transparency
- Direct object communication
- Dynamic Invocation Interface.

But CORBA also has some disadvantages. Since CORBA is only a standard specification, to build application on it, we need to buy a specific CORBA ORB product; for example: Orbix Enterprise, Orbix Standard, Orbacus (the CORBA for the Enterprise) [13], they are expensive and consume a lot of computer resources. It is worthwhile for developing large and complex application but seems over budget for our small plotter.

An ORB is much more sophisticated than alternative forms of client-server middleware, including the traditional Remote Procedure Calls (RPCs), Message-Oriented Middleware (MOM), database store procedures, and peer-to-peer services. [14] On the client side, the user needs to download an ORB at run time. This may affect the performance of the application. And different CORBA products don't have good compatibility, for example, VisiBroker client may not communicate properly with an Orbix server, although it maybe supposed to be a standard. But our goal is to build a cheap, simple, efficient and flexible plotter. So, we didn't choose CORBA as our middleware.

DCOM

DCOM is the distributed extension to COM (Component Object Model) that builds an object remote procedure call (ORPC) layer to support remote objects.[15] COM is a component based development model for Windows environment. A component is a reusable piece of software in binary form (as opposed to source code), that can be plugged into other components from other vendors with little effort.

COM has its roots in OLE (Object Linking and Embedding), shipped with Windows 3.1. The idea of OLE was to provide an improved mechanism for dealing with compound

documents and enabling things like smart documents that could embed or link Excel spreadsheet into a word document. OLE is mainly a library of predefined interfaces and default implementations for some of them. Over the years, OLE faded into the background and, with the release of Windows NT 4.0 in 1996, COM took center stage. COM supports component interaction on the local machine, both in one address space (process) and across separate address spaces. DCOM provides the same functionality but across machines over a network.

In COM, the request and responses are delivered via the Lightweight Remote Procedure Calls (LRPC). DCOM supports clients and servers residing on separate nodes (remote server). In DCOM, the mechanism for request and response remains the same, except DCOM replaces LRPC with Object-Oriented RPC (ORPC) that uses network protocol, developed upon the base of DCE remote procedure calls from OSF. In both COM and DCOM, remote method calls are synchronous. DCOM has many benefits [16]:

- Large User Base and Component Market
- Binary Software Integration

  - Large-scale software reuses (no source code)

  - Cross-language software reuse.

- On-line software update.

  - Allows updating a component in an application without recompilation, relinking or even restarting.

- Multiple interfaces per object Wide selection of programming tools available, most of which provide automation of standard code.

But DCOM is tightly integrated with the Windows platform and also heavily relies on its registry system. COM has a security model. In order to obtain a portable application, we decided not to use DCOM as our middleware.

Java RMI

Java RMI is a mechanism that allows one to invoke a method on an object that exists in another address space. The other address space could be on the same machine or a different one. The RMI mechanism is basically an object-oriented RPC (Remote Procedure Call) mechanism. [17]

Java RMI relies on a protocol called the Java Remote Method Protocol (JRMP). Each Java/RMI Server object defines an interface, which can be used to access the server object outside of the current Java Virtual Machine (JVM) and on another machine's JVM. The interface exposes a set of methods, which are indicative of the services offered by the server object.

For a client to locate a server object for the first time, RMI depends on a naming mechanism called an RMIRegistry that runs on the Server machine and holds information about available Server Objects. A Java/RMI client acquires an object reference to a Java/RMI server object by doing a lookup for a Server Object reference and invokes methods on the Server Object as if the Java/RMI server object resided in the client's address space.

Java/RMI server objects are named using URLs and for a client to acquire a server object reference, it should specify the URL of the server object as you would with the URL to a HTML page. Since Java/RMI relies on Java, it can be used on diverse operating system platforms from mainframes to UNIX boxes to Windows machines to handheld devices as long as there is a Java Virtual Machine (JVM) implementation for that platform [16].

Advantages of RMI [18]

- Support seamless remote invocation on objects in different virtual machines.
- Support callbacks from servers to applets.
- Integrates the distributed object model into the Java language in a natural way while retaining most of the Java language's object semantics.
- Makes writing reliable distributed applications as simple as possible.
- Preserves the safety provided by the Java runtime environment.
- Single language.
- Free.

We can see that, RMI has many advantages. However, it also has limitation. When the application is a scaled system, [19] which means the system is consis of a certain amount of small models instead of one big-size model, the performance becomes slow. But, since our system is just a small system; this limitation is not seen to be important here. So, we choose RMI as our middleware.

Another disadvantage about RMI is that, RMI is a pure Java solution and is specifically designed to operate in the Java environment. Using RMI as middleware, all client components and server components must be written in Java language. But in our program, we want to reuse some C++ code. This code comes from one old project, which is a standalone 2-D Graph Plotter application which can be used to plot simple mathematical expressions. [4] This creates a problken that we solve by using a Java technology called Java Native Interface (JNI).

2.1.4 Java Native Interface

The Java Native Interface (JNI) is the native programming interface for Java that is part of the JDK. Programs written using the JNI are completely portable across all platforms. [20]

The JNI allows Java code that runs within a Java Virtual Machine (JVM) to operate with applications and libraries written in other languages, such as C, C++, and assembly. In addition, the Invocation API allows you to embed the Java Virtual Machine into your native applications.

We already have a parser written in the C++ programming language. Since C++ is more efficient for calculating than Java, we don't want to write another passer under Java programming environment. So we want to make the C++ parser accessible to Java applications. We use the JNI to write native methods to handle those situations so that we don't need to rewrite the parser.

JNI serves as the glue between Java and native applications. The following diagram shows how the JNI ties the C side of an application to the Java side in our program.

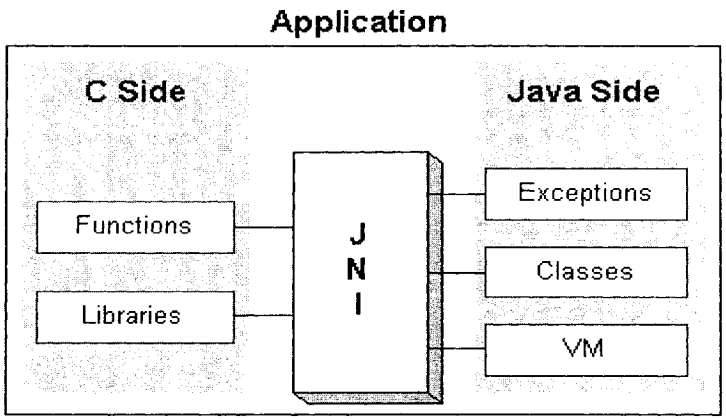

Figure 2.2: JNI ties an application to the Java

2.2 System Architecture

Figure 2.3 show the system architecture of this plotter application. We can see that, this system consists of three major subsystems: the client subsystem, the server subsystem and the parser subsystem. The details of each component are described as follows:

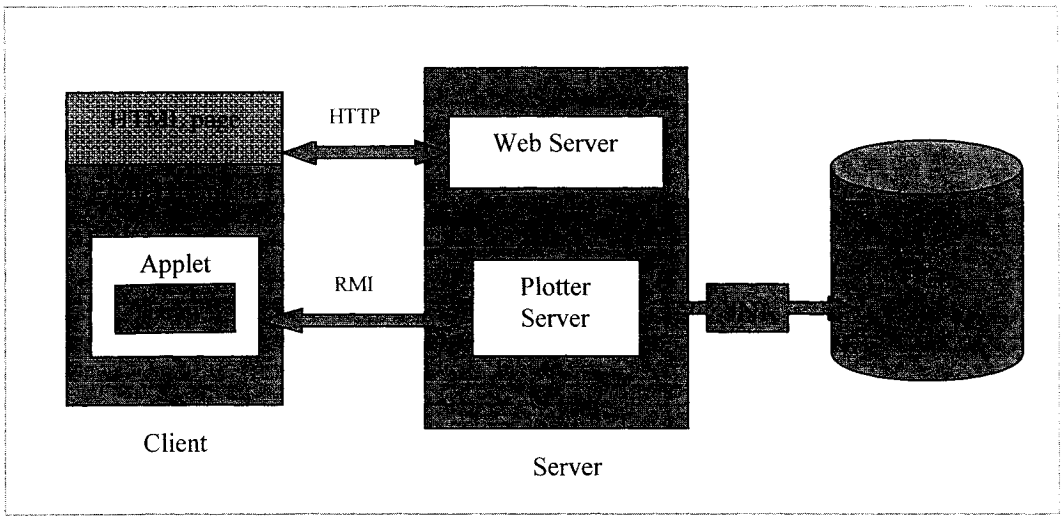

Figure 2.3: System Architecture Diagram

HTML page

The HTML page is where the user can see and get access to the system. We can say that it is the real user interface. User uses it to input formulas and to get the required 3D curves and surfaces. The plotter client applet is embedded in this HTML page. This file is called "plotter.html" and can be accessed by most popular web browsers.

Plotter3DApplet

Plotter3DApplet is the core part of the Client subsystem. The client subsystem is the client side component of this application. It consists of a class called *Plotter3DApplet* that introduces a view onto which the 3-D curves and surfaces can be plotted. And class *Plotter3DMorphBehavior* and a series of auxiliary functions like *createSceneGraph(), actionPerformed, getCurrInput, updateData...* accompany the main class to finish the desired work. The class *Plotter3DApplet* needs to be executed in a web browser. We use Java RMI technology to communicate between the client and the web server. Here, we

also provide one function *getTextAreaDebug()* to trace compiling and running. Trace information can be seen in the text area of the user interface.

HTTP

Our client html page use HTTP (Hypertext Transfer Protocol) to find server address. HTTP is a protocol with the lightness and speed necessary for a distributed collaborative hypermedia information system. [21] It is a generic stateless object-oriented protocol, which may be used for many similar tasks such as name servers, and distributed object-oriented systems, by extending the commands, or methods used.

HTTP allows a set of methods to be used to indicate the purpose of a request. This protocol is a request-response protocol. A client sends a request to the server in the form of a request method, followed by a message containing request modifiers, client information, and so on. The server responds with a status line, including the message's HTTP protocol version and a success or error code, followed by a message containing server information.

RMI

RMI is an extension of Remote Procedure Call (RPC). [22] RPC abstracts the communication interface to the level of a procedure call. RPC, however, does not translate well into distributed object system, where communication between program-level objects residing in different address space is needed. On the other hand, RMI

directly integrates a distributed object model into the Java language. To accomplish this RMI introduces local proxy objects to manage the invocation on a remote object.

The proxy object on the client side is called a stub. The proxy on the server side is called skeleton. The client and the stub are in the same Java virtual machine. The server and the skeleton are in another Java virtual machine. When the stub receives a message from the client, it forwards the request and marshals the parameters across the network to the skeleton. The skeleton performs a similar function on behalf the server, it obtain server's responds and return to client. Figure 2.4 shows how client subsystem and server subsystem communicates via stub and skeleton:

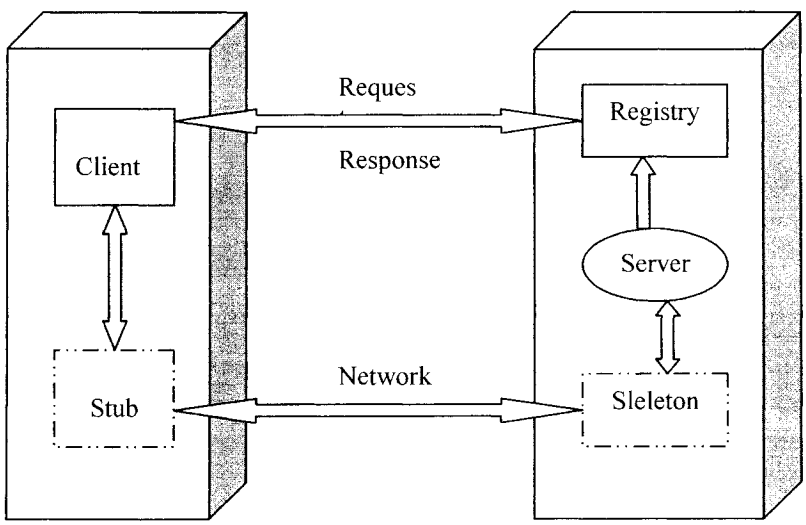

Figure 2.4: communicates via stub and skeleton

To generate the codes for stub and skeleton, we need to define a remote interface for the server. Then we use a tool provided by Java to generate the proxy objects. RMI also

provide a registry services that runs on the server machine. When a client wants to use a remote object, it asks by name for the reference to the remote object. The registry scans its database and returns a reference to the requested object. Once the client obtains the object reference, it can call a method implemented by the server objects as if the server were running on the same Java virtual machine.

Plotter Server

PlotterServer is the server-side subsystem; it extends Java class *UnicastRemoteObject* and implements class *PlotterServerInterface*. This server can access request from client. In order get the response, the plotter server calls the passer DLL to parse and evaluate the input formula, and then puts the result into class *Result*. *PlotterServerInterface* defines the RMI interface.

JNI

The JNI interface declares the native methods *ParseExpr()*, *SetVariable()*, *EvalExpr()*, *ClearExpr()*... to allow the server to access the paser.

Parser DLL

The parser we use here is old code used by Meng Cai 's projects [4]. As mentioned in section 1.3 of this report, the reuse of the parser and parser DLL component reveals many benefits.

1. It saves development time and eases the development significantly. To reuse code, all we need to do is just understand parser interface, and use the parser DLL directly.

2. Since the parser module has already been tested extensively, the reuse of the parser makes the current application more reliable.

3. Since the parser component is clearly separated from the system, the maintenance is made easier. Also, the parser component can be extended independently when needed.

4. Java is powerful for internet programming. But C++ is more powerful for parsing and other mathematical tasks. By reusing the parser and parser DLL, we can take advantages of both programming languages.

2.3 Object Model and Class Diagram

The object model shows the static structure of this plotter application and organizes into workable pieces. The model describes all independent object class and their relationships to each other. Information for our object model comes from the requirement statement and our knowledge of the application domain.

2.3.1 Class diagram

The class diagram in Figure 2.5 shows classes and the relationships between classes.

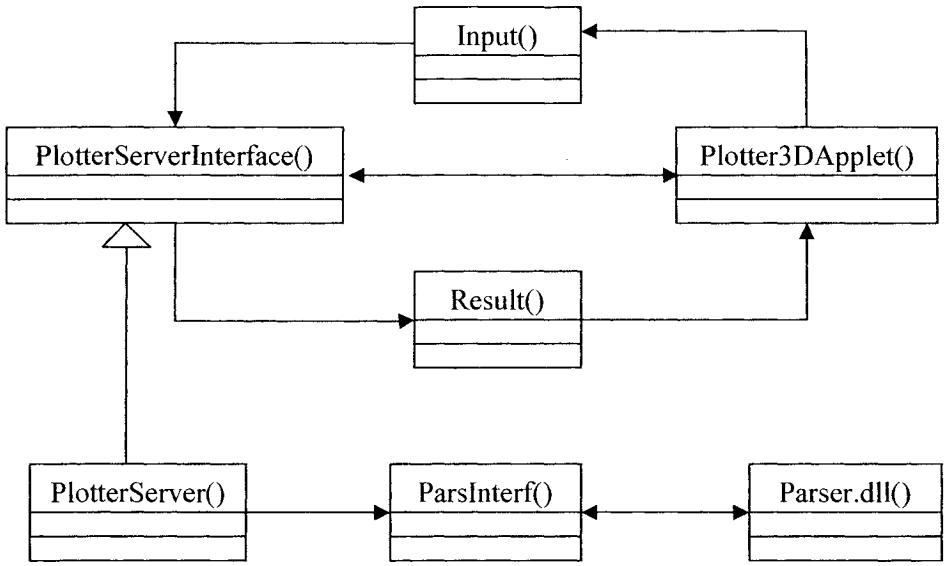

Figure 2.5: Class diagrams of Modules in Major Components

3 PART III. USER INTERFACE

3.1 User interface

Figure 3.2 show the user interface of the 3-D plotter. It is embedded in the browser internet explorer.

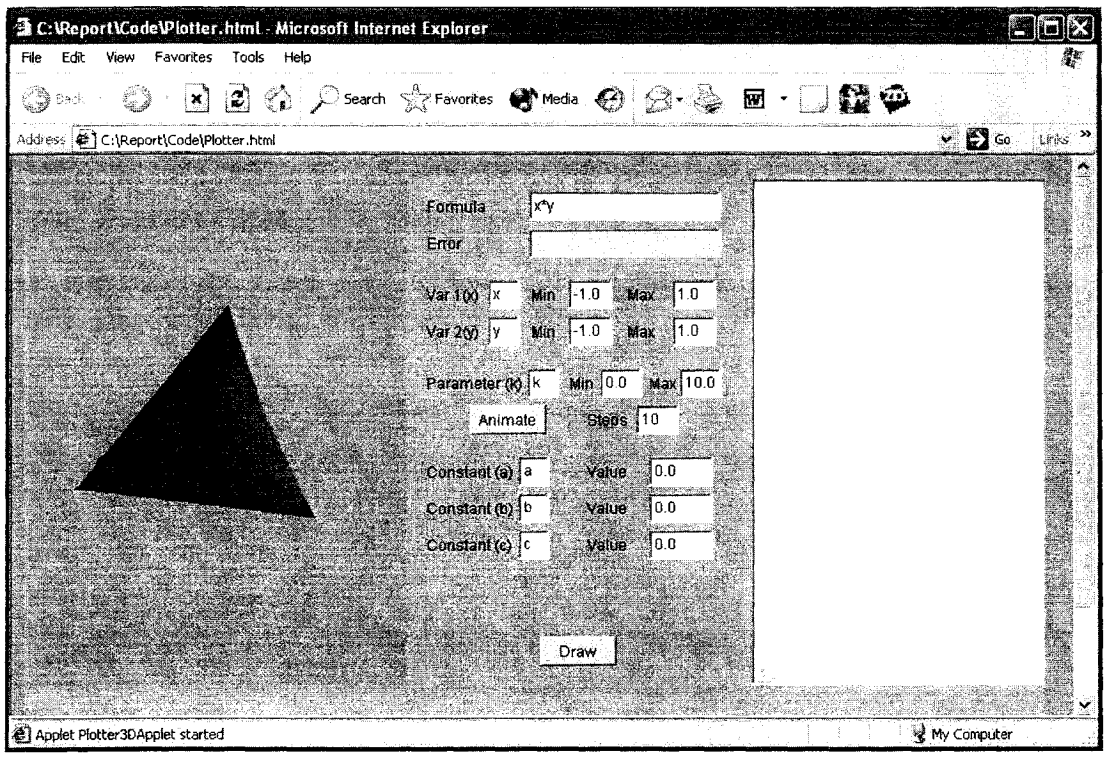

Figure 3.1: user interface plotter.html

As we can see from Figure 3.1, this user interface consists three parts:

View window

The left part is a view window in which three-dimensional curves and surfaces are displayed. When the application is started, this area is blank. After the user has input the desired expression, and clicked the draw button, the corresponding 3D graph appears with lighting effects.

Debug widow

The right part is a text area used as a debug widow. We use some trace message when we debug the program. When we running the program, trace information appear in this area. This window is for the developer and maintainers.

Input/ Output window

The middle part is the facility form built by the Java Abstract Window Toolkit (AWT). It includes labels, text areas and buttons. We can see that it provides the familiar look-and-feel interface.

- The top line is the formula field, where the user inputs exressions.

- Following it is the text field for error message. For example, if we miss a parameter in the formula, the error message "formula is missing parameter" is appears in this area.

- Next, we have two input text field for variable identifiers. The minimum and maximum value of the identifiers can also be changed from here.

- The fifth line is the parameter k and its range. These fields are used in animation function.

- The sixth line includes the button "animation" which is used to activate the animation function. The test area "step" is use to control animation precision.

- Under the *animation* button, there are areas for three constants, this means that we can use three difference constants in our formular. Their values can be input from here. The default values of the constants are zero.

- On the bottom of this part is the *draw* button. When the user clicks this button, the client function is activated, the client parses the formula and checks the syntax, and the required 3D color curves or surfaces appear in the view window.

3.2 Scenario to draw 3D curves

This scenario describes how the user inputs a formula and how the system responds, including the output of appropriate error message. It consists of six steps:

1. The user accesses the internet, and opens the plotter.html page.

2. The user inputs the expression including the variables, constants, parameter and their value and ranges.

3. The user clicks the draw button.

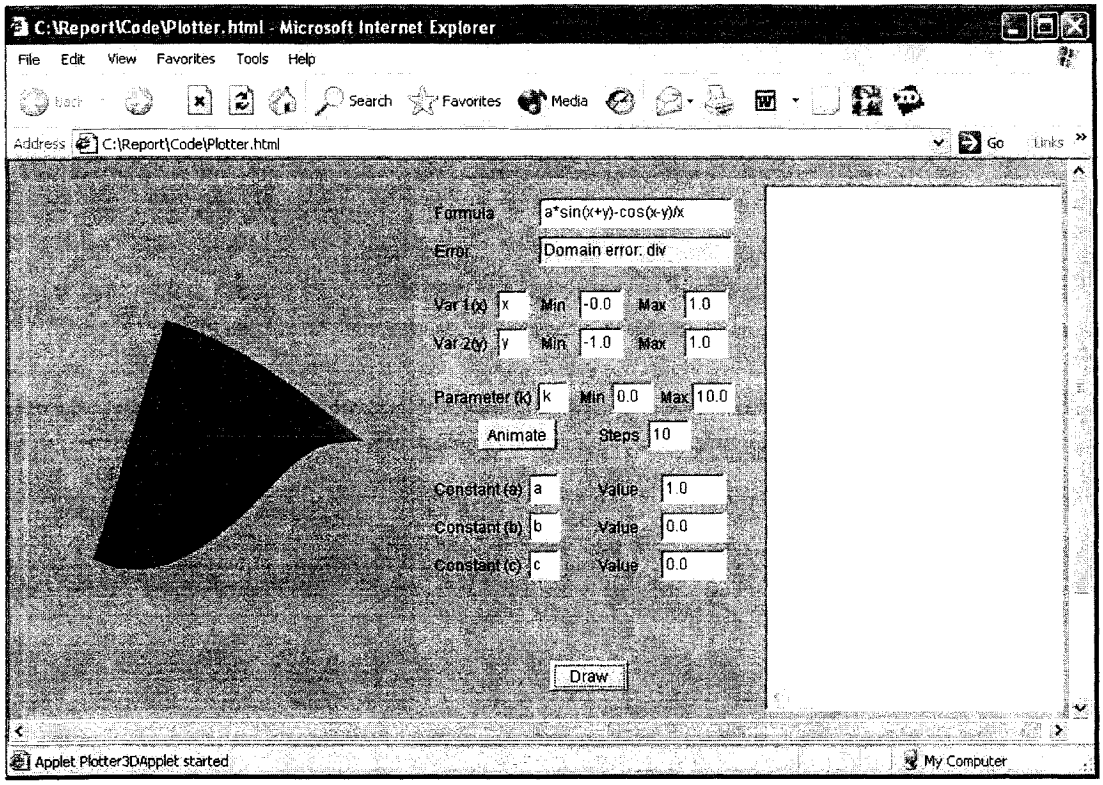

Figure 3.2: The system responds an error message

4. The parser checks the expression syntactically and builds a parse tree. If the expression is not syntactically correct, an error message appears on the error text area.

5. If the expression is syntactically correct, the parser assigns values to the variables in the parse tree and returns the value of the whole parsing tree.

6. The client plotter3Dapplet draws the 3D curve or surface on the view window.

3.3 Scenario to process animation

This scenario describes how the user interface accepts the animation request and outputs the correspondent messages.

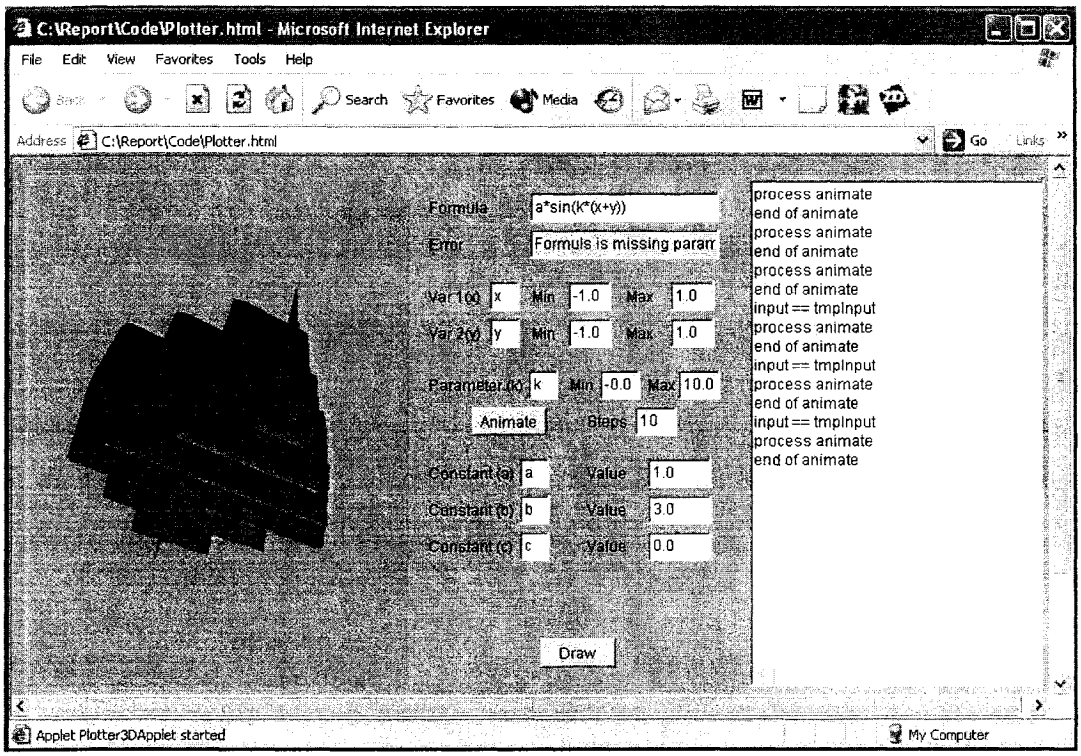

Figure 3.3: The system process the animation

1. The user accesses the internet, and opens the plotter.html page.

2. The user inputs the expression including the variables, constants, parameter and their value and ranges.

3. The user clicks the animation button.

4. The parser checks the expression syntactically and builds a parse tree. If the expression is not syntactically correct, an error message appears in the error text area.

5. The trace information appears on the debug window.

6. If the expression is syntactically correct, the parser assigns values to the node in the parse tree, and returns the value of the whole parse tree.

7. The client plotter3Dapplet display required dynamic 3D color curves according to parameter ranges.

4 PART IV. IMPLEMENTATION

4.1 Using Java 3D to implement the interface

The graphical user interface of the plotter is implemented using Java 3D API. As we describe in the chapter 2.1.2, Java 3D is a very good way to add interactive 3D graphics to a web page. Java 3D uses a "Scene Graph" concept, which breaks a 3D application into a "content branch" and a "view branch". These branches are further divided up into a tree structure which allows related items to be grouped together. This, along with other object oriented constructs is much easier and faster to write than detailing how each line or set of points is to be drawn.

The Scene Graph is a fundamental concept to Java 3D, and its purpose is to get the developer to think about the geometry of the objects in the virtual universe and how the universe is composed. It is a tree-like structure, which goes along with the languages' object-oriented paradigm.

The view branch specifies how a Viewing Platform is set up so that the virtual universe can be seen on the user's desired display device. The view branch enables Java 3D to render a single or multiple screens from a single universe, so the user can view this universe through a single screen, multiple screens, or other immersive display devices. The content branch describes the virtual objects and the parameters that specify how they

look, behave, interact, and how they are arranged in the virtual universe. The content branch can contain thousands of nodes that describe elaborate and complex 3D universes.

Objects in the Scene Graph can be tied together through TransformGroups, which have a Transform3D object. A Transform3D object encapsulates the typical 4x4 matrix that specifies a scaling, rotation, translation, and sheer to be performed. All the items in the subgraph of a TransformGroup object are modified on the screen when the TransformGroup is modified. This makes it easy to modify entire groups of objects independently from the rest of the universe.

An example of a simple scene graph is shown in Figure 4.1 [8].

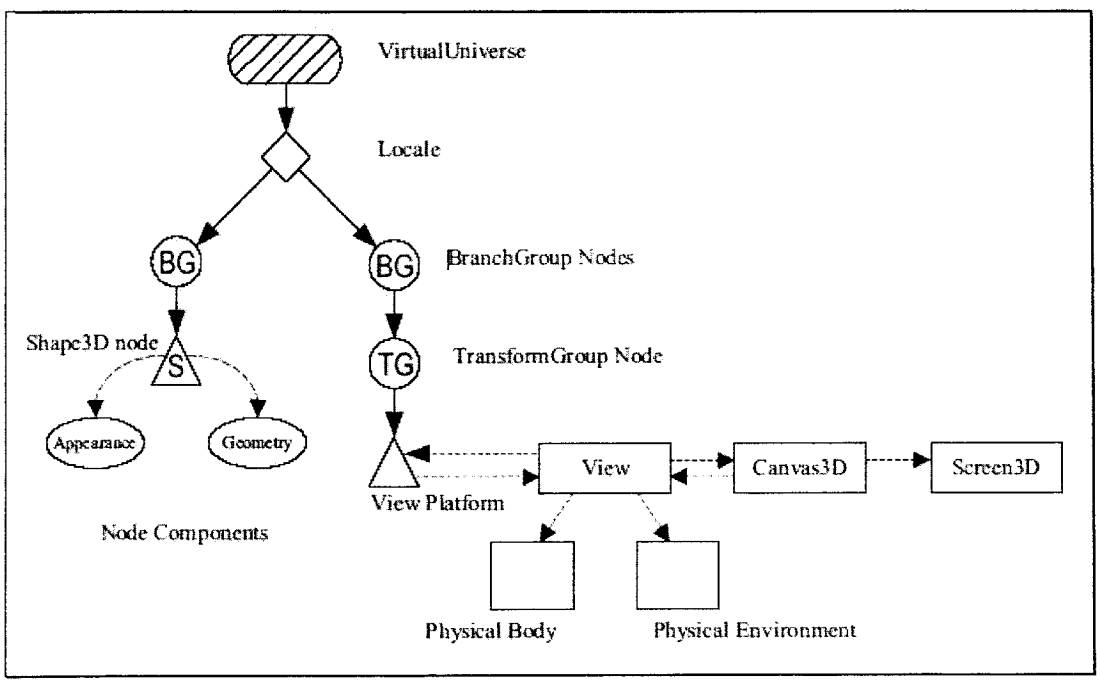

Figure 4.1 Scene Graph diagram

SceneGraphObject has two subclasses: Node and NodeComponent. The subclasses of SceneGraphObject are the building blocks that are assembled into scene graphs. The basic outline of Java 3D program development can be described by a simple recipe presented as below:

1. Create a Canvas3D object
2. Create a SimpleUniverse object which references the earlier Canvas3D object
    a. Customize the SimpleUniverse object
3. Construct content branch
4. Compile content branch graph
5. Insert content branch graph into the Locale of the SimpleUniverse

We use this recipe in our plotter client program, as below:

```
public void init() {
                    … …
                    Canvas3D canvas3D = new Canvas3D(config);      //1.
                    getPanelPicture().add("Center", canvas3D);

                    // SimpleUniverse is a Convenience Utility class
                    simpleU = new SimpleUniverse(canvas3D);        //2.

                    // This moves the ViewPlatform back a bit so the
                    // objects in the scene can be viewed.
                    simpleU.getViewingPlatform().setNominalViewingTransform();

                    BranchGroup bgSceneGraph = createSceneGraph();        //3.
```

```
        bgSceneGraph.compile();                    //4.
        simpleU.addBranchGraph(bgSceneGraph);      //5.

        getButtonAnimate().addActionListener(this);
        ... ...                    :
}
```

Lighting

Our 3D graphic has the lighting property. Lighting in 3D graphics is not trivial, but the Java 3D API makes creating good lighting much simpler. Java 3D provides classes such as Light, AmbientLight, DirectionalLight, PointLight, and SpotLight. In additoin, there are many ways to set up the lights. Parameters include attenuation, direction, concentration, influencing bounds, intensity, scoping, and position. Objects have a transparency attribute, which determines whether light passes through them. Lighting is important to the realism of a 3D scene, and the Java 3D API provides all the necessary means to light up a virtual universe.

A number of steps are necessary to enable lighting for visual objects in the virtual universe. In addition to creating and customizing light objects, each light object must be added to the scene graph and have a bounds object specified. Each visual object to be shaded must have surface normals and material properties. These requirements including:

1. Light Source specification

    a. set bounds

    b. add to scene graph

2. Visual object

a. normals

b. material properties

In our program, we set two different directional lights on 3D curves, light 1 is red and light 2 is green:

```
// Set up the global lights
Color3f light1Color = new Color3f(1.0f, 0.0f, 0.0f);
Vector3f light1Direction  =  new Vector3f(0.0f, 0.0f, 3.3f);
Color3f light2Color  =  new Color3f(0.0f, 1.0f, 0.0f);
Vector3f light2Direction  =  new Vector3f(-1.0f, -1.0f, -0.8f);
Color3f ambientColor  =  new Color3f(0.1f, 0.1f, 0.1f);

DirectionalLight light1  =  new DirectionalLight (light1Color,
                                        light1Direction);
light1.setEnable(true);
light1.setInfluencingBounds(bounds);
objRoot.addChild(light1);

DirectionalLight light2  =  new DirectionalLight(light2Color,
                                        light2Direction);
light2.setEnable(true);
light2.setInfluencingBounds(bounds);
objRoot.addChild(light2);

AmbientLight ambientLightNode = new AmbientLight(ambientColor);
ambientLightNode.setEnable(true);
ambientLightNode.setInfluencingBounds(bounds);
objRoot.addChild(ambientLightNode);
```

Then we set material properties. We create an appearance in function createSceneGraph():

```
… …
Appearance look = new Appearance();
Material material = new Material();
look.setMaterial(material);
PolygonAttributes polyAttr = new PolygonAttributes();
polyAttr.setCullFace(PolygonAttributes.CULL_NONE);
polyAttr.setBackFaceNormalFlip(true);
look.setPolygonAttributes(polyAttr);
morphObj.setAppearance(look);
… …
```

Then we calculate the normal in the function *updateData*, see appendexA.1

```
… …
Vector3f normal = new Vector3f();
int normalIndex = 0;
… …
// calculate the normal
if (j < input.xResolution-1) {
        vertex2.x = ((Float)(results.xCoordinate.elementAt(j+1))).floatValue();
        vertex2.y = ((Float)(results.yCoordinate.elementAt(i))).floatValue();
        vertex2.z = ((Float)(zValue.elementAt(zIndex))).floatValue();
```

```
line1.sub(vertex2, vertex0);
line2.sub(vertex1, vertex0);
normal.cross(line1, line2);
normal.normalize();
... ...
```

Animation

Animations in Java 3D are implemented using Behavior objects. Behaviors are a way for users to interact with objects, and for objects to interact with each other. Like a call back function, Behaviors allow the programmer to specify what actions should be taken when certain events or combinations of events take place. Behaviors can be made to trigger on time-related functions, mouse and keyboard events, changes in other objects, and a variety of other occurrences. A behavior is made up of a wakeup criterion and a set of one or more actions. The wakeup criterion determine what set of events will trigger the behavior, and the actions determine what will be performed when the behavior is triggered. Behaviors can be used for collision detection, reacting to the proximity of the user's view platform, handling mouse events, and other useful things.

The Java 3D API provides a number of classes useful in creating animations without having to create a new class. One set of animation classes are known as interpolators. An interpolator is a special type of behavior. It is used to modify objects as time progresses. The starting and ending values for a transform (texture, light, etc.) and how the value will

change over time is specified. The transition can be done only once, or the interpolator can cycle through the change repeatedly.

Interpolator classes change various visual attributes in the virtual world. However, there is no interpolator to change the geometry of a visual object. This is exactly what the Morph Class does. A Morph object creates the geometry for a visual object through interpolating from a set of GeometryArray objects.

A Morph object stores an array of GeometryArray objects. GeometryArray is the superclass of TriangleArray, QuadStripArray, IndexedLineStripArray, and TriangleFanArray. The individual GeometryArray objects each completely defines one complete geometric specification for the visual object including color, normals, and texture coordinates. The GeometryArray objects can be thought of as key frames in an animation, or more properly, as constants in an equation to create a new GeometryArray object. In addition to the array of GeometryArray objects, a Morph object has an array of weight values – these are the variables in the equation. Using the GeometryArray objects and the weights, a Morph object constructs a new geometry array object using the weighted average of the coordinate, color, normals, and texture coordinate information from the GeometryArray objects. Changing the weights changes the resulting geometry. All that is required to use a Morph object is to create the array of GeometryArray objects and set the weighting values.

The following steps are needed to use a Morph object.

1. Create an array of GeometryArray objects

2. Create a Morph object with ALLOW_WEIGHTS_WRITE

3. Assemble the scene graph, including adding children to target Switch object(s)

We implement this in the function *createSceneGraph()* .First we create GeometryArray[], an array of GeometryArray objects:

```
geometryArray = new GeometryArray[Input.MAXPARASTEPS];
for (int kk = 0; kk< Input.MAXPARASTEPS; kk++) {
    geometryArray[kk] = createGeomArray();
}
```

Then we create a morph object:

```
Morph morphObj = new Morph(geometryArray);
morphObj.setCapability(Morph.ALLOW_GEOMETRY_ARRAY_WRITE);
morphObj.setCapability(Morph.ALLOW_GEOMETRY_ARRAY_READ);
morphObj.setCapability(Morph.ALLOW_WEIGHTS_WRITE);
... ...
```

We implement the Morph object in the class *Plotter3DMorphBehavior*:

```
class Plotter3DMorphBehavior extends Behavior{
    ... ...
    public Plotter3DMorphBehavior(Morph morph) {
```

```
            targetMorph = morph;
            alpha = new Alpha();
            weights = new double[Input.MAXPARASTEPS];
            startAnimate = new WakeupOnBehaviorPost(this, 1);
            duringAnimate = new WakeupOnElapsedFrames(0);
    }
    ... ...
```

Interaction

Java 3D API introduced the GeometryUpdater interface which, along with BY_REFERENCE geometry, provides the ability to change the geometry data at runtime. With the GeometryUpdater interface, an application programmer can produce any type of animation that depends on changing geometry information.

The GeometryUpdater object modifies the geometry when invoked. The behavior object schedules the invocation of the GeometryUpdater object in a GeometryUpdater application.

Using a GeometryUpdater object for dynamic geometry requires creating BY_REFERENCE geometry with the appropriate capabilities, creating a GeometryUpdater class and instantiating an object of it, and creating a custom Behavior class and instantiating an object of it. All this is not as difficult as it may seem.

The GeometryUpdater object modifies the geometry when invoked. The behavior object schedules the invocation of the GeometryUpdater object in a GeometryUpdater application.

```
public void updateData(Geometry geometry) {
... ...
... ...
} // end updateData(Geometry)
```

The detail of the updateData(Geometry) see the appendex A1.

4.2 Implementation of the midware RMI

To communicate with the server, plotter client use RMI technology to locate a server. RMI depends on a naming mechanism, called an RMIRegistry, which runs on the Server machine and holds information about available Server Objects. The plotter client acquires an object reference to a Java RMI server object by doing a lookup for a server object reference and invokes methods on the server object as if the Java RMI server object resided in the client's address space.

```
static final String URL = "rmi://localhost/plotter";
... ...
PlotterServerInterface server = null;
... ...
setName("Plotter3DApplet");
... ...
try     {
```

```
        server = (PlotterServerInterface) Naming.lookup(URL);
        }
catch (Exception ex) {
... ...
```

4.2.1 Defining the functions of the remote class as an interface

A remote object is an instance of a class that implements a *Remote* interface. The remote interface declares each of the methods that we would like to call from other Java virtual machines (JVMs). In our program, we define an interface which is inherited from Remote call PlotterServerInterface. The remote interface extends the java.rmi.Remote interface. This function gets the parameter from the client side which includes the varables and parameter ranges and calculates the values of corresponding value for the formula and transfer back to the client side.

```
public interface PlotterServerInterface extends Remote
{
        public Results evaluate(Input input) throws RemoteException;
}
```

4.2.2 Writing the implementation and server classes

A "server" class is the class which has a *main* method that creates an instance of the remote object implementation and binds that instance to a name in the RMIregistry.

```
public class PlotterServer extends UnicastRemoteObject implements
PlotterServerInterface
```

```
{
        static final String SERVERNAME = "//localhost/plotter";
        ParsInterf parsInt;
        ......
}
```

4.2.3 Implementaton of the remote interface

The plotter server implements the remote class interface *PlotterServerInterface*, It provide the application logic of the plotter. The implementation class can extend a remote class like the java.rmi.server.UnicastRemoteObject. By extending this, the class can be used to create a remote object that:

- Uses RMI's default sockets-based transport for communication
- Runs all the time

```
public interface PlotterServerInterface extends Remote{...}
```

4.2.4 Defining the constructor for the remote object

The constructor for a remote class provides the same functionality as the constructor for a nonremote class: it initializes the variables of each newly created instance of the class, and returns an instance of the class to the program which called the constructor.

```
public PlotterServer() throws RemoteException{
                parsInt = new ParsInterf();
... ...
}
```

4.2.5 Providing an implementation for each remote method

The implementation class for a remote object contains the code that implements each of the remote methods specified in the remote interface. Arguments to, or return values from, remote methods can be any data type for the Java platform, including objects, as long as those objects implement the interface java.io.Serializable. In PlotterServer class, we implement the remote method evaluate.

```
public Results evaluate(Input input) throws RemoteException {
        Results result = new Results();
        ......
        ......
        return result;
}
```

4.2.6 Creating and installing a security manager

One of the most common problems one encounters with RMI is a failure due to security constraints. The *main* method of the server first needs to create and install a security manager. A security manager needs to be running so that it can guarantee that the classes that get loaded do not perform operations that they are not allowed to perform. If no security manager is specified, no class loading, by RMI clients or servers, is allowed, aside from what can be found in the local CLASSPATH. A Java program may specify a security manager that determines its security policy. When we run the server program, we start the security manager.

java -cp c:\Report\WebRoot -Djava.rmi.server.codebase=http://localhost/

-Djava.security.policy=c:\Report\code\policy PlotterServer

4.2.7 Creating one or more instances of a remote object

The main method of the server needs to create one or more instances of the remote object implementation, which provides the service. The constructor exports the remote object, which means that once created, the remote object is ready to accept incoming calls.

```
PlotterServer server = new PlotterServer();
```

4.2.8 Registering the remote object

For a client to be able to invoke a method on a remote object, that caller must first obtain a reference to the remote object.

```
Naming.rebind(SERVERNAME, server);
```

JNI

Using Java Native Interface for Java programs is a multi-step process. After writing the Java program, we create a Java class that declares the native method; this class contains the declaration or signature for the native method. It also includes a main method which calls the native method. For example, in our program, we declare

```
public native int ParseExpr(String jexpr);
```

```
public native int SetVariable(String jname, double value);

public native double EvalExpr()

public native void ClearExpr();
```

Then, we compile the Java class that declares the native method and the main method. Next, we generate a header file for the native method using javah with the native interface flag -jni. Once we have generated the header file we have the formal signature for our native method.

```
rem javah -classpath c:\Report\Code -jni -o c:\Report\Code ParsInterf

javah -jni ParsInterf
```

Then, we write the implementation of the native method in the programming language of our choice, such as C or C++. For this project, we reuse the old C++ parser. Finally, we compile the header and implementation files into a shared library file. We include the PaserInterf.java in appendix A-2.

5 PART V Conclusion and Future Works

5.1 Conclusion

In brief, we can conclude the following from this project:

1) This 3-D plotting utility was developed to run on distributed computing system, this means the program is running on single server machine, many clients can access this tool from their client computer through internet; they don't need to install the system themselves. So the resource is shared.

2) This plotting utility is user friendly, the user interface has good look-and-feel, and users can learn to use it very easily. If they are familiar with the World Wide Web environment, they do not need any special knowledge to use this tool.

3) This plotting utility is efficient and has good viewing appearance. After testing, all well-formed expressions consisting of identifiers (with one or more letters), constant numbers, function names (support only built in functions) and operators can be parsed in a short time, and the user can get pretty, colored three-dimensional curves and surfaces rapidly.

4) This plotting utility provided code reuse. The component technology is based on object-oriented model, these abstraction models can be reuse easily, and provide the

possibility of extension. The parser used in this application is also old code written in C++.

5) This utility is cost effective. We choose to use free software and open technology to develop this application. And these softwares and technologies are very popular and were well tested. The development cost is very small.

5.2 Future work

This project can be extened by many ways:

1) It is possible for Java users to provide special rendering, sound effects, printing capacity, and other creative options that can be generated by a designer.

2) More functionality like texture property, scaling and rotation can be added to this application.

3) More property can be added to existing functionality. For example, for lighting property, a different shading model could be added.

4) Security and authentication features can also be added to this application.

6 References

[1] http://www.wolfram.com/products/mathematica/index.html

[2] http://www.mathworks.com. The web page of MatLab software

[3] http://www.math.utah.edu/lab/ms/maple/maple.html

[4] Meng Cai, A PLOTTING TOOL FOR INTERNET BASED ON CLIENT/SERVER COMPUTING MODEL, Master's Major Report, Concordia University, 2001

[5] Anh Phong Tran, 2-D Graph Plotter: A Tool for Plotting Functions, Master's Major Report, Concordia University, 2000

[6] http://www.opengl.org/

[7] http://mesa3d.sourceforgenet/

[8] Getting Started with the Java 3D™ API, Dennis J Bouvier, © 1999-2001 Sun Microsystems, Inc. 2550 Garcia Avenue, Mountain View, California 94043-1100 U.S.A

[9] http://www.javaworld.com/javaworld/jw-01-1999/jw-01-media.html, 3D graphics programming in Java

[10] Sun Microsystems, *"Java Remote Method Invocation Specification"*, October 1997,

http://java.sun.com/products/JDK/1.1/docs/guide/rmi

[11] Gerald Brose, Andreas Vogel, and Keith Duddy, Java Programming with CORBA, third Eddition, OMG Press, 2000

[12] P.Emerald Chung, Yennun Huang, Shalini Yajnik, Deron Liang, Joanne C.Shih, Chung-Yih Wang, and Yi-Min Wang, DCOM and CORBA Side by Side, Step by Step, and Layer by Layer

[13] Component based development – *KPN Research* http://www.serc.nl/espinode/evenementen/19990525_CBD/presentaties/Nieuwenhuis/sld001.htm

[14]http://www.cse.msu.edu/~patilabh/research/Final_Report.pdf Comparison of Middleware Technologies - CORBA, RMI & COM/DCOM Abhishek Patil, Rajesh Korde, Kapil Sabharwal, Michigan State University

[15] Microsoft White Paper, *"Windows DCOM Architecture"*, 1998

[16] Gopalan Suresh Raj, A Detailed Comparison of CORBA, DCOM and Java/RMI http://my.execpc.com/~gopalan/misc/compare.html

[17] http://www.iona.com/products/

Getting started with RMI – http://java.sun.com/products/jdk1.1/docs/guide/rmi/

[18] Java, RMI and CORBA, A white paper prepared by David Curtis, Copyright © 1997 by Object Management Group, http://java.ittoolbox.com/Documents/

[19]http://www.brooks.af.mil/AFRL/HED/hedr/reports/handbook/chp7-2-6.htm

[20] Java Native Interface, The Programmer's Guide and Specification, 1/e, Sheng Liang, 1999, ISBN 0-201-32577-2

[21] http://www.w3.org/Protocols/HTTP/HTTP2.html

[22] RMI Tutorials - http://www.ccs.neu.edu/home/kenb/com3337/rmi_tut.html

[23] http://java.sun.com/docs/books/tutorial/index.html The Java Tutorial, Trail: Java Native Interface

[24] A New Approach to Distributed Computing White Paper http://www.cetus-links.org/oo_java_rmi.html

[25] http://www.swtech.com/java/native/ Java Native Methods

7 APPENDIX A: Selected Source Code

7.1 A-1: updataData(Geometry)

```
/********************** ListVehiclesByChosenModel.ui.tcl ******************/

public void updateData(Geometry geometry) {
        System.out.println("updateData enter");
        try {
        TriangleStripArray triangleStripArrayUpdate = (TriangleStripArray)geometry;
        float[] coord = triangleStripArrayUpdate.getCoordRefFloat();
        float[] normals = triangleStripArrayUpdate.getNormalRefFloat();

        Vector zValue = (Vector)(results.zCoordinate.elementAt(zCoordinateIndex));
        int zIndex = 0;
        int i = 0;
        int j = 0;
        int base = 0;
        Vector3f vertex0 = new Vector3f();
        Vector3f vertex1 = new Vector3f();
        Vector3f vertex2 = new Vector3f();
        Vector3f line1 = new Vector3f();
        Vector3f line2 = new Vector3f();
        Vector3f normal = new Vector3f();
        int normalIndex = 0;
        for (i = 0; i < input.yResolution-1; i++) {
                base = i * 6 * input.xResolution;
                for (j = 0; j < input.xResolution; j++) {
                        vertex0.x = coord[base + 6*j+0] =
                        ((Float)(results.xCoordinate.elementAt(j))).floatValue();
```

```
vertex0.y = coord[base + 6*j+1] =
((Float)(results.yCoordinate.elementAt(i))).floatValue();
vertex0.z = coord[base + 6*j+2] =
((Float)(zValue.elementAt(zIndex))).floatValue();

vertex1.x = coord[base + 6*j+3] =
((Float)(results.xCoordinate.elementAt(j))).floatValue();
vertex1.y = coord[base + 6*j+4] =
((Float)(results.yCoordinate.elementAt(i+1))).floatValue();
vertex1.z = coord[base + 6*j+5] =
((Float)(zValue.elementAt(zIndex+input.xResolution))).floatValue();

zIndex++;

// calculate the normal
if (j < input.xResolution-1) {
vertex2.x =  ((Float)(results.xCoordinate.elementAt(j+1))).floatValue();
vertex2.y = ((Float)(results.yCoordinate.elementAt(i))).floatValue();
vertex2.z = ((Float)(zValue.elementAt(zIndex))).floatValue();

line1.sub(vertex2, vertex0);
line2.sub(vertex1, vertex0);
normal.cross(line1, line2);
normal.normalize();
normals[normalIndex++] = normal.x;
normals[normalIndex++] = normal.y;
normals[normalIndex++] = normal.z;

line1.sub(vertex0, vertex1);
line2.sub(vertex2, vertex1);
normal.cross(line1, line2);
normal.normalize();
normals[normalIndex++] = normal.x;
normals[normalIndex++] = normal.y;
```

```
                    normals[normalIndex++] = normal.z;
                } else { // last two vertex normal in the colume
                    vertex2.x = coord[base + 6*j-3];
                    vertex2.y = coord[base + 6*j-2];
                    vertex2.z = coord[base + 6*j-1];

                    line1.sub(vertex1, vertex0);
                    line2.sub(vertex2, vertex0);
                    normal.cross(line1, line2);
                    normal.normalize();
                    normals[normalIndex++] = normal.x;
                    normals[normalIndex++] = normal.y;
                    normals[normalIndex++] = normal.z;

                    line1.sub(vertex2, vertex1);
                    line2.sub(vertex0, vertex1);
                    normal.cross(line1, line2);
                    normal.normalize();
                    normals[normalIndex++] = normal.x;
                    normals[normalIndex++] = normal.y;
                    normals[normalIndex++] = normal.z;
                }
            }
        }
        // adjustment for the normal between the two column
        Vector3f normal1 = new Vector3f();
        Vector3f normal2 = new Vector3f();
        int base1 = 0;
        int base2 = 0;
        for (i = 0; i < input.yResolution-2; i++) {
            base1 = i * 6 * input.xResolution;
            base2 = (i+1) * 6 * input.xResolution;
```

```
            for (j = 0; j < input.xResolution; j++) {
                    normals[base1 + 6*j+3] = normals[base2 + 6*j+0];
                    normals[base1 + 6*j+4] = normals[base2 + 6*j+1];
                    normals[base1 + 6*j+5] = normals[base2 + 6*j+2];
                    }
            }
} catch (Exception exception) {
            getTextFieldError().setText("updateData error");
            exception.printStackTrace();
            }

} // end updateData(Geometry)
```

7.2 A-2: Code for Java Native Interface

```
/************************** ParserIntef.java ***************************/

class ParsInterf {
   public native int ParseExpr(String jexpr);
   public native int SetVariable(String jname, double value);
   public native double EvalExpr()
         throws IllegalArgumentException;
   public native void ClearExpr();

   static {
      System.loadLibrary("parser");
   }
```

```
    public int parseExpr(String expr)
    {
        return ParseExpr(expr);
    }

    public int setVariable(String nm, double val)
    {
        return SetVariable(nm, val);
    }

    public double evalExpr()
    {
        double val = 0.0;
//      try {
            val = EvalExpr();
            return val;
//      } catch (Exception e) {
        //System.out.println("In Java:\n\t" + e);
        //return 0.0;
//    }
    }

    public void clearExpr()
    {
        ClearExpr();
    }
}
```

7.3 A-3: Code for HTML file

```
/***************************** Plotter.html ****************************/


<!--"CONVERTED_APPLET"-->
<!-- CONVERTER VERSION 1.0 -->
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
WIDTH = 800 HEIGHT = 500  codebase="http://java.sun.com/products/plugin/1.44/jinstall-12-
win32.cab#Version=1,4,0,mn">
<PARAM NAME = CODE VALUE = "Plotter3DApplet.class" >
<PARAM NAME = CODEBASE VALUE = "http://127.0.0.1" >

<PARAM NAME="type" VALUE="application/x-java-applet;jpi-version=1.4">
        <COMMENT>
        <EMBED type="application/x-java-applet;jpi-version=1.4"
        WIDTH = 800 HEIGHT = 500
        CODE =  "Plotter3DApplet.class"
        CODEBASE =  "http://127.0.0.1"
        pluginspage="http://java.sun.com/j2se/1.4/download.html">
        <NOEMBED>
        </NOEMBED>
        </EMBED>
        </COMMENT>
</OBJECT>

<!--
<APPLET  CODE = "Plotter3DApplet.class" CODEBASE = "http://127.0.0.1" WIDTH = 800
HEIGHT = 500 >
<PARAM NAME = CODE VALUE  = "Plotter3DApplet.class" >
<PARAM NAME = CODEBASE VALUE  = "http://127.0.0.1" >
<PARAM NAME = "type" VALUE ="application/x-java-applet;version=1.1.2">
</XMP>
```

```
</APPLET>
-->
<!--"END_CONVERTED_APPLET"-->
```