National University of Singapore

Web-based Electromagnetic simulation

Fan Yun

(M.Eng., Northwestern Polytechnic of University, P.R. China)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2003

Acknowledgement

This thesis has become possible due to the generous and ongoing support of many people.I would like to take this opportunity to express my deepest and sincere appreciation to them.

First and foremost, I wish to thank my supervisors, Dr. Lee Er-Ping and Associate Professor Li Le-Wei, for their contribution and support throughout the entire course of the project. Special thanks to my immediate supervisor Dr. Lee Er-Ping for his patience and enthusiastic guidance.

Secondly, I would like to thank to Mr. Li Yong-Lin and Ms. Wu Fang for their advice and guidance on the programming. I would like to thank to Dr. Yuan Wei-Liang for his support in the study of FDTD.

Thirdly, I would like to thank to Ms. Jin Hong-fang and Ms. Sun Yan for their suggestions and help in the study of electromagnetic simulation.

Last but not least, I would like to thank my father, mother and family for their continuous encouragement and support during the entire study and project.

Abstract

Electromagnetic (EM) simulators have traditionally been used as back-end analysis tools to verify the results of circuit simulators. They work by accurately modeling the physical effects in a variety of structures ranging from transmission lines to planar circuits.

Most of electromagnetic simulation packages, such as Ansoft HFSS and FDTD packages, are available on UNIX workstations running at X Windows and Windows NT and 2000. However, modeling and simulation of complex electromagnetic problems should run at high capacities and expensive supercomputers, as well as their maintenance.

With the development of web resources, it is evident how important it is to design simulation tools which could take advantage of the Web. This thesis discusses the methodology and development of web-based electromagnetic simulation system, which enables users to simulate complex electromagnetic problems and visualize the results at different platforms at different location. Web applications techniques, including socket communication, multithreading programming, communication among applet, servlets, and VRML techniques, are discussed in the following chapters.

The system takes advantage of the popular programming language—Java, which is portable across very different platforms on the World Wide Web. Servlets is used in the web server, and applet and HTML are used in the user end. The interface between Java and VRML is studied to interactively visualize the simulation results of a vrml file remotely.

The socket technique is used to implement the communication between the web server and high performance computer. In order that multiple users can access the high performance computer and do the electromagnetic simulation synchronously, multithreading technique is used to achieve multi-access to the electromagnetic simulation packages.

The interface between Java and VRML is also studied and discussed in the thesis. The Java-VRML interaction takes place through the EAI, which gives the functionality to extend the features of VRML by adding the power of the programming language Java. Integrating VRML and Java through EAI can achieve the web-based cooperative electromagnetic simulation.

List of Contents

List of Figures

List of Acronyms

EM	Electromagnetic
FEM	Finite Element Method
FDTD	Finite Difference Time Domain
HFSS	High Frequency Structure Simulator
GUI	Graphical User Interface
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
EAI	External Application Interface
VRML	Virtual Reality Modeling Language
HTTP	Hypertext Transfer Protocol
JVM	Java Virtual Machine
RMI	Remote Method Invocation

Chapter 1 Introduction

This chapter begins with a brief overview of computational electromagnetics methods used in the electromagnetic simulation, and reviews the previous development done on this topic. The motivation and objectives of this research project are then presented. These are followed by a brief highlight on the scope and outline of various topics discussed in this thesis.

1.1 Overview on Computational Electromagnetics Methods

With the rapid growth in communication and information systems, over the past years, it has created an increasing demand for higher and higher speed electronics. Hence, the electrical performance analysis and simulation, EMC and signal integrity at high IC speeds become exceptionally crucial issues to design engineers. It, therefore, is significant to accurately and efficiently simulate the large and complex electromagnetic problems for electrical designers jointly with other engineers to accomplish a common simulation at one platform.

The Maxwell's equations are essential for the modeling of electromagnetic fields. All current electromagnetic numerical methods are developed, based on solving the Maxwell's equations. Since 1960, many numerical methods had been developed to solve the Maxwell's equations accurately and quickly. These methods can be classified as frequency domain ones, such as the moment method (MM), the finite element method (FEM), the boundary element method (BEM); and time domain

methods, e.g., the finite differential time-domain (FDTD). The following paragraphs give a brief description on the two typical methods implemented in this thesis.

1.1.1 Finite Element Method (FEM)

Finite element method (FEM) is a flexible full-wave method for electromagnetic computation [1]. FEM is employed to solve the variational equations, which are derived from the differential form of Maxwell's equations in frequency domain. In the FEM, the computational domains are meshed into elements with simple shapes and the electromagnetic fields are expanded using the shape functions associated with these elements. Then, the variational equations are discretized into a matrix equation. There are node-based and edge-based shape functions for FEM. Node-based shape functions have been used extensively in civil and mechanical engineering applications, while edge-based shape functions are often used for solving electromagnetic problems. For the open-region problems, it is necessary to truncate the FEM mesh with an appropriate absorbing boundary condition, such as the perfectly matched layers (PML). The finite element method is also hybridized with the boundary intega1 equation to solve the open-region problem. The whole problem is divided into an inner and an outer region, then, the FEM is used to formulate the fields in the inner region and the boundary integral equation is used to represent the fields in the outer region. The tangential fields across the interface boundary between inner and outer regions are enforced continuity to obtain a coupled system of equations. The advantage of this hybrid method over the absorbing boundary condition is that it can reduce the number of unknowns produced due to the introduction of the absorbing boundary condition. The matrix arising in FEM is sparse and symmetry, which is able

to reduce the computing time and memory requirements for solving the matrix equation.

1.1.2 FDTD and its Improvement

The finite-difference time-domain (FDTD) method is also a full-wave method for electromagnetic computation [2]. FDTD method is a simple and elegant algorithm to discretize the differential form of Maxwell's equations in time domain. In the FDTD analysis, the whole computational region is divided by grids, and an electric field grid offset both spatially and temporally from a magnetic field grid is used to obtain update equations that yield the present fields throughout the computational domain in terms of the past fields. Similar as FEM, to solve open-region problems with FDTD method, an absorbing boundary condition must be applied to truncate the computational domains. There have been several types of absorbing boundary condition developed for FDTD method, such as Mur's ABC which is based on the differential equation and PML ABC which is material ABC. FDTD approach is an efficient and accurate method for solving EMC problems. Unfortunately, FDTD method suffers from the high computational cost since it needs volume meshing and time-stepping. Recently, parallel FDTD algorithm had been developed to solve this problem.

There are many improvements on the standard FDTD method, such as the finite-difference frequency-domain (FDFD) [2] and the finite-volume time-domain (FVTD) methods [3]. For the standard FDTD analysis, frequency dependent parameters must be calculated by Fourier transform of the time domain results. Due to the fixed time step the mesh has to be uniform. To accurately model sharp discontinuities, a very

small mesh size has to be employed, which leads to a very extensive computational effort. At the same time, for the resonant structures, small time step is required to reach a sufficient resolution in the frequency domain, which is very time consuming. The finite differential frequency domain method is proposed to overcome this problem. It solves the Maxwell's equations in frequency domain. Therefore it can avoid the Fourier transform and give the frequency dependent parameters directly. It had been successfully app1ied to eigenvalue problems for calculating the propagation characteristics of cavity resonators, and the chip interconnections [4]. For the standard FDTD analysis, the computing errors are usually caused by the staircasing when the scatterer surface is not one of the known coordinate surface. To eliminate the drawback existing in the standard FDTD analysis, the finite volume time domain method is developed, where the tangentnormal grids that conform the scatterer surface are introduced. These body grids intersect among themselves and also intersect with a rectangular grid that is used away from the scatterer surface. Of course, FVTD method requires more memory and computing time than the standard FDTD method.

1.2 Introduction

1.2.1 Backgrounds of Web-based Electromagnetic Simulation

As mentioned above, electromagnetic (EM) simulation has been considered as a key step in the design of electromagnetically critical elements ranging from simple microwave stripline to large complex systems. There are numerous electromagnetic simulation packages available for virtual electronic products deign, optimization and characterization. These packages are able to accurately model the physical effects in various structures ranging from transmission lines to planar circuits.

However, there exist a number of essential problems. When simulating a complex and detailed model, huge CPU and computing resources are required to obtain accurate results within a reasonable time. Consequently, high capacity HPC resources are necessary in order to perform fast and accurate simulation.

Furthermore, the globalization of electronics industry has compelled the use of Internet for virtual design, simulation and analysis. This requires a group of simulation models, residing on different machines connected to the web, which are analyzed by designers globally and accessible for designers on independent platforms. Therefore, the portability of applications across different platforms should be achieved.

However, the portability of applications across different platforms is difficult and expensive for an application with a truly portable graphical user interface (GUI). Portability requires a certain amount of conformance of operating systems, programming languages and tools. In most cases, a GUI is limited to a certain environment, for example, to the Windows platform. In addition, a new version of any of the underlying ingredients may already break the original GUI.

Moreover, majority of the existing electromagnetic software packages are only available on UNIX workstations running X Window. It can be seen obviously from the problem stated above, that the current electromagnetic simulation packages have the limitation to meet global design requirements, and, a new solution must be developed.

1.2.2 Motivation

There are mainly two approaches to achieve accurate and affordable results within a reasonable time: firstly to run the simulation on a very costly, high performance super-computer, or secondly to share the simulation on a cluster of small-sized PCs.

Given this scenario, it is evident how important is to design simulation tools which could take advantage of the Web or computer network. So it is necessary that a group of simulation models, residing on different machines attached to the Internet, collaborate with each other to accomplish a common task and are accessible for designers or users with different platforms. Developing web-based interactive programs to simulate the complex electromagnetic problems and to visualize the results in different platforms at different location is a challenging task.

It is more difficult if the applications must be portable across very different platforms, such as UNIX and Microsoft Windows, but portability of application is necessary in the web-based electromagnetic simulation systems. However, a more appealing and effective approach is to use Java language and associated techniques. Java has gained enormous popularity as a programming language for the World Wide Web but Java also offers similar advantages for GUI applications outside the World Wide Web. The latest Java release, with its rich set of GUI tools, is being ported to virtually all platforms, and true to its "write once, run everywhere" slogan. It requires no special maintenance for differing platforms, and that translate into cost saving.

With implementation of Internet programming capabilities offered by Java, as well as new web standards such as XML, a cost effective solution can be achieved to develop the web-based electromagnetic simulation systems.

1.2.3 Objectives

The aim of this research is to study the software architectures for electromagnetic simulation in order to design and implement them on the Internet. In this thesis, two simulation packages HFSS, a FEM based High Frequency Structure Simulator, and FDTD, are integrated into a web-based environment by using the Java and associated technique.

The basic idea is to re-define the software implementation of electromagnetic simulation packages (e.g. FDTD simulation package) based on the web applications, according to the object-oriented paradigm. The web-based electromagnetic simulation system should allow the users to access the applications wherever from the Web through a Web browser without the aid of client or local applications. Users can subscribe and remotely access the high-end resources via Internet through a low-end computer with a browser. It makes electromagnetic simulation available at anytime, anyplace and on any device. In addition, the complete compatibility of the application with the Web will allow distributing the execution of the simulation on cluster of net connected computers. The most difficult part is the correct design of the tool, since only a correct and rigorous analysis and object-orientation design will guarantee the positive features mentioned above.

1.3 Dissertation Outline

A brief summary of the thesis organization is given as follow.

Chapter 1 reviews the computational electromagnetics methods used in electromagnetic simulation, typically EMC simulation methods. This chapter also introduces the background, current development, and problems of electromagnetic simulation packages. Finally, a summary of the motivation and objective of the web-based electromagnetic simulation system is presented.

Problems faced by electromagnetic simulation packages are described in Chapter 2. The relevant methodologies and techniques are used to overcome these problems given in Chapter 2 as well as part of Chapter 3. In the rest of Chapter 3, web-based HFSS simulation system is introduced, and its architecture and development are given. In this chapter, it also presents the three-tier distributed computing architecture for extending EM simulation resources from high-end servers to low-end desktop computers. Another example of web-based EM simulation system development is given in Chapter 4. After the web-based FDTD system is introduced, and problems in the development and the consequence solving methodologies are described.

The interface between Java and VRML is presented in Chapter 5. Through the interface of EAI, the integration of Java and VRML is available. The vrml file—.wrl produced by EM simulation can be interactively visualized in the internet. The conclusion and the future work are summarized in Chapter 6.

Chapter 2 Basic Techniques Used in Web-based EM Simulation

In this chapter, a concept of web-based EM simulation system and its general implementation will be presented. Basic techniques, including Java techniques, object serialization and architecture of system, used in the web-based EM simulation are described in details.

2.1 Conception

The massive growth of the Internet and the World-Wide Web leads to a completely new way of looking at the development of engineering simulation systems. In order to make EM simulation packages easily and conveniently to the users, and to run the simulation on the independent platform as well as the designers be able to cooperate together to accomplish a common simulation, it is necessary to develop the web-based EM simulation system.

To perform the complicated EM simulation, it requires high performance computers. In general, EM simulation systems run on the high performance computing platforms, and are independent of web servers, for example running at a UNIX server or an application server. A web server accomplishes communications between designers and EM simulation packages. To implement a given EM simulation, the following procedures are applied;

- Designers at the user end need a browser to input parameters. Then the web server transmits the input data to EM simulation packages that shall perform the simulation at an application server.

- According to the parameters input by the designers, EM simulation packages accomplish the simulation and produce the final data results, then return them to the web server.

- According to the special requirements from the designers, the web server post-processes the data results from the application server, and transmits them to end users. Finally, the data is visualized on the monitor of end users.

The general model of web-based EM simulation system is illustrated in Fig 2.1. The general network architecture is shown in the Fig 2.2.

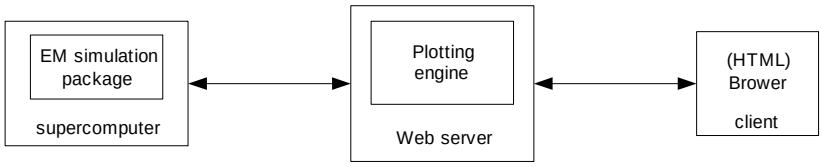

Figure 2.1: The diagram of general web-based EM simulation system

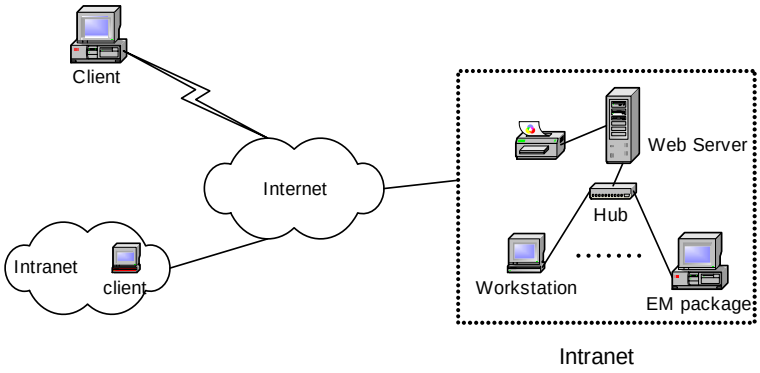

Figure 2.2: The network architecture of general web-based EM simulation system

With the features and advantages of Java [5], the Java programming languages, serialization and its associated techniques are selected for the web-based design language, while multithreading technique is used to let EM simulation systems accessible to multiple designers. These techniques are described in this chapter and part of Chapter 3. Java components and relative techniques used in the EM simulation systems here are described in the following sections.

2.2 Java Techniques

To live in the world of electronic commerce and distribution, Java must enable the development of secure, high performance, and highly robust applications on multiple platforms in heterogeneous and distributed networks. The design requirements of Java are driven by the nature of the computing environments in which software must be deployed, so in the EM simulation system Java techniques are used and their characteristics are described as follows.

2.2.1 Object Oriented

Java is designed to be object oriented from the ground up where object technology has finally found its way into the programming mainstream after a gestation period of thirty years. The needs for distributed and client-server based systems coincide with the encapsulated, message-passing paradigms of object-based software. To properly function within the increasingly complex and network-based environments, the programming systems must adopt object-oriented concepts. To meet the needs, Java programming language is able to provide such a clean and efficient object-based development environment, so that many EM simulation packages can be developed as graphical user interface (GUI) and web-based applications by using the means of Java and its relative techniques. With the aids of the advanced Java techniques, EM simulation packages can offer an easy and convenient way for designers' access.

2.2.2 Robust and Secure

Java is designed for creating highly reliable software. It provides extensive compile-time checking, followed by a second level of run-time checking. Java is designed to operate in distributed environments, which means that security is of paramount importance. With security features designed into the language and run-time system, Java lets programmers construct applications that cannot be invaded from outside. In the networked environment, applications written in Java are secure from intrusion by unauthorized code attempting to get behind the scenes and create viruses or invade file systems.

2.2.3 Architecture Neutral and Portable

Java is designed to support applications that will be deployed into heterogeneous networked environments. In such environments, applications must be capable of executing on a variety of hardware architectures. Within this variety of hardware platforms, applications must execute atop a variety of operating systems and interoperate with multiple programming language interfaces. To accommodate the diversity of operating environments, the Java compiler generates bytecodes—an architecture neutral intermediate format designed to transport code efficiently to multiple hardware and software platforms. The interpreted nature of Java solves both binary distribution problem and version problem. The same Java language byte codes will be able to run at any platform.

Architecture neutrality is just one part of a truly portable system. Java takes portability a stage further by being strict in its definition of the basic language. Java puts a stake in the ground and specifies the sizes of its basic data types and the behavior of its arithmetic operators. Programs are the same on every platform—there are no data type incompatibilities across hardware and software architectures.

The architecture neutral and portable language environment of Java is known as the Java Virtual Machine. It's the specification of an abstract machine for which Java language compilers can generate code. The specific implementations of the Java Virtual Machine for specific hardware and software platforms, therefore, can provide the concrete realization of the virtual machine. The Java Virtual Machine is primarily based on the POSIX interface specification—an industry standard definition of a portable system interface. Implementing the Java Virtual Machine on a new

architecture is a relatively straightforward task as long as the target platform meets basic requirements such as support for multithreading.

It is noticeable that the portable feature of java can provide a unique function where EM simulation packages are able to run at independent platform, which meets EM simulation's requirements on platforms and their performances. This unique feature also makes Java as the main web-based development programming language, and it is also the main reason that Java is selected as the programming language in the web-based EM simulation development.

2.2.4 Interpreted, Threaded, and Dynamic

The Java interpreter can execute Java bytecodes directly on any machine to which the interpreter and run-time system have been ported. In an interpreted environment such as Java system, the link phase of a program is simple, incremental, and lightweight. We can benefit from much faster development cycles—prototyping, experimentation, and rapid development are the normal case, versus the traditional heavyweight compile, link, and test cycles.

Modern web-based applications, such as the web-based EM simulation system, typically need to do several things at the same time. While implementing an EM simulation or computing, many users working with the web-based EM simulation system can run a simulation package concurrently. Java's multithreading capability provides the means to build the applications with many concurrent threads of activity. Multithreading, which will be described in details in chapter 3, thus results in a high degree of interactivity to the end user.

Java supports multithreading at the language level with the addition of sophisticated synchronization primitives: the language library provides the Thread class, and the run-time system provides monitor and condition lock primitives. At the library level, moreover, Java's high-level system libraries have been written to be thread safe: the functionality provided by the libraries is available without conflict to multiple concurrent threads of execution.

While the Java compiler is strict in its compile-time static checking, the language and run-time system are dynamic in their linking stages. Classes are linked only when required. New code modules can be linked in on demand from a variety of sources, even from sources across a network. In the case of the HotJava browser and similar applications, interactive executable code can be loaded from anywhere, which enables transparent updating of applications. The result is on-line services that constantly evolve; they can remain innovative and fresh, draw more customers, and spur the growth of electronic commerce on the Internet.

2.3 Protocol

Java programs that communicate over the network are typically concerned only with the application layer, and use the classes java.net package, which provide a system-independent network communication. However, to decide which Java classes your programs should use, it is necessary to understand how transmission control protocol (TCP) and user datagram protocol (UDP) differ.

2.3.1 Transmission Control Protocol (TCP)

TCP is a connection-based protocol that provides a reliable flow of data between two computers.

In more detail, TCP provides a point-to-point channel for applications that require reliable communications. The Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), and Telnet are all examples of applications that require a reliable communication channel. The order in which the data is sent and received over the network is critical to the success of these applications. When HTTP is used to read from a URL, the data must be received in the order in which it was sent. Otherwise, users end up with a jumbled HTML file, a corrupt zip file, or some other invalid information.

2.3.2 User Datagram Protocol (UDP)

User Datagram Protocol (UDP) is a protocol that sends independent packets of data, called datagrams, from one computer to another with no guarantees about arrival.

More specifically, the UDP protocol provides for communication that is not guaranteed between two applications on the network. UDP is not connection-based like TCP. Rather, it sends independent packets of data, called datagrams, from one application to another. Sending datagrams is much like sending a letter through the postal service: The order of delivery is not important and not guaranteed, and each message is independent to other.

In the web-based EM simulation system, the communication that occurs between the client and the server must be reliable and no data can be dropped. It must arrive on the client side in the same order in which the server sent it. That is, the accurate and reliable parameters and data transmission are required, so TCP and relative java classes are used in the development of web-based EM simulation system.

2.4 Java components

2.4.1 Java Servlets

The interface between the web server and the simulation tools has ever mainly been based on CGI (Common Gateway Interface) scripts. However, in CGI scripts, when a new process is created by each request on the server that loads an interpreter, in turn the interpreter runs another process for each invocation of a module simulator. Consequently, the server bears a significant load, the greater the number of users connected to the system the worse the server gets.

Java Servlets are server-side components that are analogous in many ways to CGI programs [10]. They handle web requests, returning data or HTML programmatically rather than from a static file. They can access databases, perform calculations, and communicate with other components such as Enterprise JavaBeans. Unlike CGI programs, however, Servlets are persistent—they're instantiated once and continually handle requests (usually many simultaneous requests) for the life of the web server. They're therefore operating at a much higher level of efficiency than CGI programs. Java Servlets technology provides web developers a simple, consistent mechanism for extending the functionality of a web server as well as for accessing existing EM

simulation packages or tools [6]. Java servlets can almost be thought of as applets that run on the server side and are intended to run as modules in conjunction with a Web server, extending its functionality in some way. The servlets communicate with web server through servlet APIs and are environment independent. Thus they are compatible with most web servers, allowing a servlet application to run across different platforms.

Performance improving is a direct consequence of developing an application in java servlets. In this way, with each request the lightweight thread is produced, rather than a new process is created with the result that once a servlet is loaded, it can be reused continually. In other words, once a servlet is invoked, it can handle further requests resulting in less use of expensive system resources and better scalability.

Servlets run within a servlet engine, usually on a web or application server. Unlike applets, servlets aren't burdened with security restrictions. Since it executes entirely on the server, it can fully utilize all the capabilities that the operating system allows.

Servlets can be used to create an easily accessible interface between clients, such as applets and web browsers, and the core enterprise applications behind them. In term of the requests at client side, there is no difference between using servlets and any other web. When the client accesses a URL, the results can be sent back through servlets. In fact, any kind of data can be sent and received via the HTTP protocol.

The server-based Java servlets is a powerful technique for developing web-based EM simulation applications run in conjunction with an EM simulation package and a

browser. It, therefore, greatly improves the availability, functionality and performance of web-based EM simulation system.

2.4.2 Java Applets

Developers use Java applets to make their Web pages more interactive and functional. Java applets are small Java programs, which are compiled to create platform independent bytecode. The resulting bytecode is generally saved on a server. The HTML tag APPLET is used to include a reference to the bytecode file. When the browser downloads the HTML page, the applet bytecode is also downloaded and the Java Virtual Machine (JVM) built-into the browser executes the bytecode that will result in the display of the applet within the browser screen. Thus, Java applets are executed on the client side.

Java applets are the essential Java programs that run within a web page [10]. They're Java classes that extend the java.applet.applet class and are embedded by reference within an HTML page, much like an image. Combined with HTML, they can make an interface more dynamic and powerful than with HTML alone. Applet can be used in an enterprise application to view or manipulate data coming from other sources on the server. For example, in the EM simulation system, an applet may be used to transmit the EM simulation parameters from users to the web server and visualize the EM simulation results from the EM simulation packages running in the high performance computer.

Besides the class file defining the Java applet itself, applets can use a collection of utility classes, either by themselves or archived into a JAR file. The applets and their

class files are distributed through standard HTTP requests and therefore can be sent across firewalls with the rest of the web page data. Applet code is refreshed automatically each time the user revisits the hosting web site, eliminating the concern of keeping the full application up to date on each client desktop to which it's been distributed.

Thanks to Java's operating system agnosticism, applets can run in any browser with a Java virtual machine (JVM). Sun's Java Plug-in software even lets programmers build pages that can take advantage of the latest JVM, instead of being restricted to whatever version of the JVM user's browser happens to have implemented.

Since applets are extensions of the Java platform, existing Java components can be reused when designers build at least a portion of web application interface with applets. As we'll see in a later example, designers can use complex Java objects developed originally for server-side applications as components of their applets. In fact, designers can write Java code that can operate as either an applet or an application.

Applets have all the functionalities of a traditional Java application, including the ability to use advanced JFC/Swing components from Sun [7]. Applets have the full graphical and user interface capability of applications, so it is convenient to be used to visualize EM simulation results. But despite their similarities, there are some key differences between applications and applets. The one that concerns us here has to do with the security constraints imposed on applets.

● Security Constraints on Applets

Applet code is served from a host web server and executed in the client's browser on the end user's machine. To prevent the proliferation of evil viral applets that could wreak havoc on unsuspecting surfers, applets are bound by security constraints that allow them to communicate only with their host server and prevent them from interacting with the end user's machine. They can't read or write from its file systems, execute programs, or examine certain sensitive environmental properties. Furthermore, applets can't create or accept foreign socket connections. Foreign here means a connection with anything other than the local server, which is the machine actually providing the applet's class files (as distinct from the host serving the HTML that references the applet).

Because of these restrictions, we must employ special strategies to communicate information to or from an applet. Several ways, which will be discussed in the chapter4 in details, give us opportunity to build real-time interactive interfaces that can use the web as their platform.

2.5 Object Serialization

The basic concept of object serialization is the ability to read and write objects to byte streams [8]. These streams can be used for sending objects over the network, for saving session state information by servlets, for sending parameters for Remote Method Invocation (RMI) calls, and for saving state information about JavaBean™ technology components, as well as many other tasks.

2.5.1 Conception

To do object serialization an ObjectOutputStream is used to save objects and an ObjectInputStream to read them back. The serialization process deals with flattening out an object tree such that all the raw data that make up an object, including all the objects referenced by that object, get saved. And, when it's time to read back an object, the original object tree graph gets recreated. Special cases like circular references and multiple references to a single object are preserved such that when the tree graph gets recreated new objects don't magically appear where a reference to another object in the tree should be.

In order for an object to support the serialization process, the class must implement the serializable interface. There are no methods in the interface; however, just adding implements Serializable to a class definition doesn't automatically make a class Serializable. All the instance variables of said class must be Serializable, also.

To serialize an instance of a class, the only things that are saved are the non-static and non-transient instance data. Class definitions are not saved. They must be available when you try to deserialize an object.

The basic process of serializing an object is as follows:

```
ObjectOutputStream oos = new ObjectOutputStream (anOutputStream);
Serializable serializableObject =...
oos.writeObject (serializableObject);
```

Then, to read the object back, deserialization, to do the reverse is:

```
ObjectInputStream ois = new ObjectInputStream (anInputStream);
```

```
Object serializableObject = ois.readObject ();
```

The other thing about serialization is overriding the readObject and writeObject methods to change the default serialization behavior. By overriding these methods we can provide additional information to the output stream, to be read back in at deserialization time. One common reason to override readObject and writeObject is to serialize the data for a superclass that is not Serializable itself.

2.5.2 Validating Streams

When serializing data to a file or sending data across a socket there is no guarantee what you receive is what was written. In the case of saving to a file, a corrupt user can try to use a byte-level editor to change around the bytes to alter the values for when the object is restored.

With serialization, a class can provide the ObjectInputStream parameter to readObject with an ObjectInputValidation interface implementer to indicate a desire to perform validation after an object is fully restored. The interface consists of a single method: public void validateObject () throws InvalidObjectException. When implemented, the class's readObject method must register the validator with the ObjectInputStream through registerValidation (ObjectInputValidation, int), the last argument of which is the validator priority, in case there are multiple (higher values are called first):

```
private void readObject (ObjectInputStream ois)
   throws ClassNotFoundException, IOException {
  ois.registerValidation (validator, 0);
}
```

2.5.3 Using ObjectStreamField

There are two ways to define which fields get streamed when an object is serialized. By default, every non-static and non-transient field is preserved. However, if a class defines an array of ObjectStreamField objects named serialFields (that happens to be private, static, and final), then you can explicitly declare the specific fields saved. The order you place fields in the array is the order in which they are written. For instance, in the following class, only the username and counter fields are serialized, not the password.

```
public class MyClass implements Serializable {
  private String username;
  private int counter;
  private String password;

  private final static ObjectStreamField []
    serialFields = {
      new ObjectStreamField ("username", String.class),
      new ObjectStreamField ("counter", int.class)
    };
  ...
}
```

2.5.4 Encrypting Serialized Objects

The Java Cryptography Extension (JCE) provides support for encryption [9]. With regards to serialization, JCE can be used to either encrypt everything along a stream with a CipherOutputStream, or seal individual objects with a Cipher through the SealedObject class. Usually, programmers can use CipherOutputStream and encrypt a

whole stream. However, SealedObject allows us to wrap any Serializable object into a sealed one for later usage, perhaps to save in a servlet session.

To demonstrate, the following example uses a CipherOutputStream to encrypt a series of objects written to disk, and then uses a CipherInputStream to decrypt the file and read the objects back. The cipher streams act just like any other filtering streams: just add the stream between the file and the object streams, e.g.

```
…
// Create Cipher
Cipher desCipher = Cipher.getInstance ("DES/ECB/PKCS5Padding");
desCipher.init (Cipher.ENCRYPT_MODE, secretKey);

// Create stream
FileOutputStream fos = new FileOutputStream ("out.des");
BufferedOutputStream bos = new BufferedOutputStream (fos);
CipherOutputStream cos = new CipherOutputStream (bos, desCipher);
ObjectOutputStream oos = new ObjectOutputStream (cos);
…
// Change cipher mode
desCipher.init (Cipher.DECRYPT_MODE, secretKey);

// Create stream
FileInputStream fis = new FileInputStream ("out.des");
BufferedInputStream bis = new BufferedInputStream (fis);
CipherInputStream cis = new CipherInputStream (bis, desCipher);
ObjectInputStream ois = new ObjectInputStream (cis);
```

The second form of encryption is sealing an object. This is done by calling the constructor for SealedObject, just passing the constructor the serializable object to seal and the Cipher to use for encryption:

SealedObject sealedObject =new SealedObject (serializable, cipher);

When it is time to unseal the object, there are three getObject methods we can use:

getObject (Cipher c)
getObject (Key key)
getObject (Key key, String provider)

Each of these unsealing methods works like the readObject method from ObjectInputStream—we have to cast the object returned to the appropriate type. Which method will be used depends upon the circumstances, though the second is probably the most frequently used, as it means the decrypter does not need to know the encryption parameters. The first version is also commonly used for when a Cipher object is shared within an application.

2.6 Architectural Options

An enterprise application that makes use of applets and servlets can conceivably be designed in more than one way. The following outline three different architectural options [10] here and describe their drawbacks and advantages.

2.6.1 A Two-tier Architecture

This architecture actually uses applets but no servlets, for despite the limitations imposed on applets by their security model, applets can use protocols such as JDBC and RMI to communicate directly with back-end information systems like databases, LDAP directories, or Enterprise JavaBeans components. This architectural model is diagrammed in Fig 2.3.

Although it seems simple, this model poses a number of problems and isn't generally a good idea. First of all, this scheme requires designers to embed all of their access information in the applet code directly. Database account names, passwords, server identifiers—everything must be coded into the applet, where the end user might be able to glean the information from the class files. Additionally, the database or whatever system you're accessing has to be on the same system as the web server that hosted the applet. This means your server has to do double duty as both a database and a web server. Typically, your back-end resources would be restricted by a firewall, but this isn't possible in this situation since the applet (acting from the client machine) must have direct access to the machine. Finally, this scheme makes pooling and clustering of web servers difficult, if not impossible.

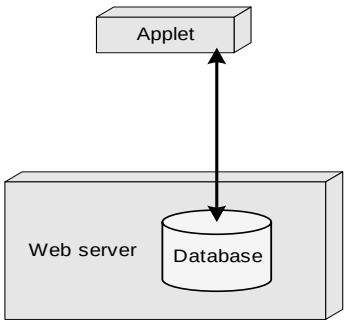

Figure 2.3: A two-tier application architecture

2.6.2 A Three-tier Architecture

A better option is to encapsulate the business of communicating with back-end resources into servlets, leaving applets to handle the front-end work only. In this architecture, sketched in Fig 2.4, servlets help overcome the security restrictions inherent in applets and control the applet's access to enterprise information systems and business logic. When a request comes into a servlet, the servlet can look up

information in a back-end database, perform calculations, or do whatever's necessary to get information on behalf of the applet or act on information from the applet. A big advantage here is that applet/servlet pairs can be deployed across a large pool of front-end web servers, all communicating with a single shared database on the back end. In addition, designing around servlets helps modularize the design, abstract the business logic behind the application, and plan for scalability.

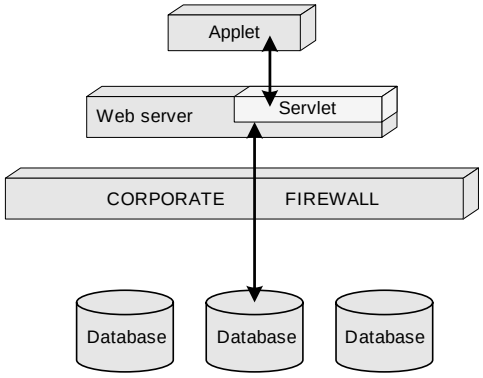

Figure 2.4: A three-tier application architecture

2.6.3 An N-tier Architecture

If designers are building an application around Enterprise JavaBeans, servlets act as the go-between. The EJB components can help further abstract the business logic, moving it outside of the servlet. In this case, an applet contacts its servlet and the servlet interfaces with the EJB component, as shown in Fig 2.5. Building an application with a hierarchy of EJB components, servlets, and applet/HTML front ends gives designers the maximum degree of flexibility and performance, though at the cost of greater complexity and expense.

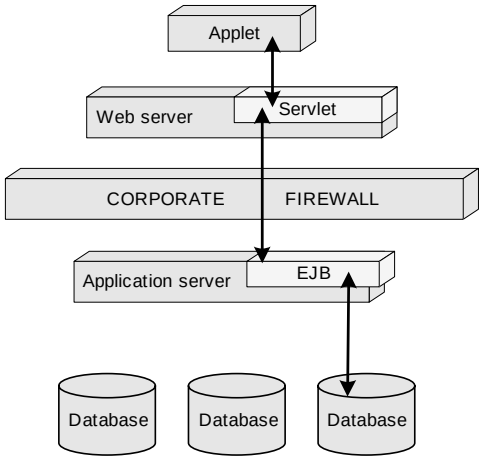

Figure 2.5: An N-tier application architecture

This chapter discusses the features, components and application architectures of Java, as well as the use of object serialization with Java language. As described above, it is clear that Java and relative techniques provide a good selection for development of the web-based EM simulation system, and its detail application will be discussed in Chapter 3 and 4 with two examples.

2.7 Conclusions

Seen in the above description, Java can achieve the development of secure, high performance, and highly robust applications on multiple platforms in heterogeneous and distributed networks. Its features, special unique portable feature, make the Java as the main web-based development programming language, and this is also the main reason that Java is selected in this project.

Instead of passing each parameter of simulation information, object serialization is able to read and write objects to byte streams. These streams can be transferred over

the network, for saving session state information by servlets, for sending parameters for RMI calls, as well as many other tasks.

Combining Java techniques with object serialization makes EM simulation packages easy and convenient to access for users. It also makes EM simulation packages run in the independent platform, and finally make designers cooperate together to accomplish a common simulation.

Chapter 3 Web-based HFSS Simulation System

Except basic techniques described in the previous chapter, another basic and important technique used in the web-based simulation system, Multithreading, is discussed. With all these basic techniques, several examples are given to illustrate the development of EM simulation system on the web. In the latter part of this chapter, the development and implementation of web-based HFSS system are presented.

3.1 Multithreading Programming

In order for multiple users to implement EM simulation simultaneously on line, the server must be capable of serving multiple clients. The answer to this problem is multithreading [11]-[15], i.e., the capabilities of server to have several "threads" of control active at the same time, with each of them responsible for the communication with a different client.

Clearly for multithreading to be viable, it must be implemented at the language level. Java supports threads at the syntactic level from its run-time system and thread objects. Its multithreading capability provides the means to build applications with many concurrent threads of activity, which results in a high degree of user interactivity [11].

3.1.1 Multithreading Basic Concept

A thread is similar to a sequential program but it is not a program. It cannot run on its own, rather, runs within a program. A thread is illustrated in Fig 3.1.

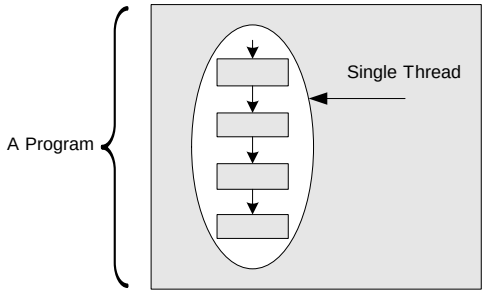

Figure 3.1: The diagram of a thread

The main advantage behind the use of multiple threads in a single program, is that they run "at the same time" and perform different tasks (or the same tasks with different parameters). Its diagram is shown in Fig 3.2.

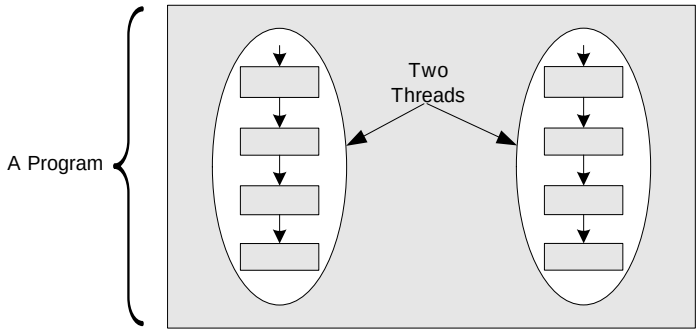

Figure 3.2: The diagram of multiple threads

A program or process contains multiple threads that execute instructions according to program code. Like multiple processes that can run on one computer, multiple threads appear to be doing their work in parallel. Implemented on a multi-processor machine, they actually can work in parallel. Unlike processes, threads share the same address space, that is, they can read and write the same variables and data structures.

In a multithreaded program, threads are obtained from the pool of available ready-to-run threads and run on the available system CPUs. The OS can move threads from the

processor to either a ready or blocking queue, in which case the thread is said to have "yielded" the processor. Alternatively, the Java virtual machine (JVM) can manage thread movement -- under either a cooperative or preemptive model -- from a ready queue onto the processor, where the thread can begin executing its program code.

Cooperative threading allows the threads to decide when they should give up the processor to other waiting threads. The application developer determines exactly when threads will yield to other threads, allowing them to work very efficiently with one another. A disadvantage is that a malicious or poorly written thread can starve other threads while it consumes all available CPU time.

Under the preemptive threading model, the OS interrupts threads at any time, usually after allowing them to run for a period of time (known as a time-slice). As a result, no thread can ever unfairly hog the processor. However, interrupting threads at any time poses problems for the program developer. The preemptive threading model requires that threads use shared resources appropriately, while the cooperative model requires threads to share execution time. Because the JVM specification does not mandate a particular threading model, Java developers must write programs for both models. In web-based FDTD system which will be described later, the preemptive threading has been used.

3.1.2 The Life Cycle of a Thread

The main stages in the life-time of a thread are shown in Fig 3.3.

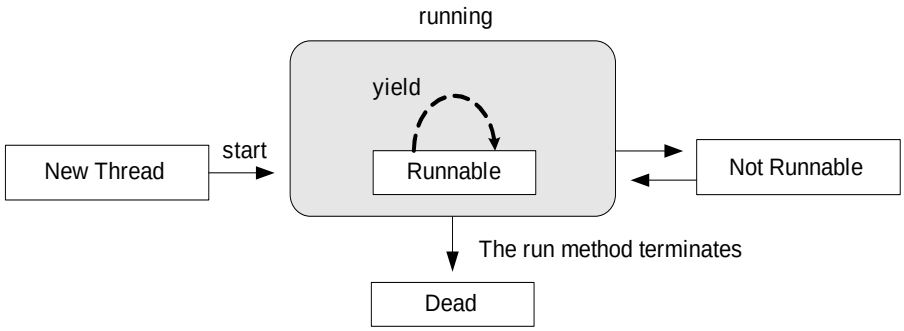

Figure 3.3: The life cycle of a thread

When a thread (object) is first created, it is in the New Thread state. It remains in this state until the thread is actually started.

Starting a thread involves a few tasks on the part of the system. The major categories to which these tasks are: creating the resources required for running the thread as a separate control sequence than the rest of the program; "scheduling" the thread (more on this soon); and, finally, running the thread.

After a thread is started, it enters its running state. This, however, does not mean that the thread is actually "active" at all points in time from there onwards. This can be intuitively understood if one considers the fact that computers with only a single processor, cannot run all "running" threads at the same time. Instead, they have to "share their time" between all threads that are in the running state. How this is achieved differs between the operating systems and processor architectures. The Java runtime makes very few promises in this respect, abiding by the lowest common denominator of thread functionality on the different systems it runs. The important thing to keep in mind from the above description is that at any given time, a "running" thread actually may be waiting for its turn in the CPU.

While in the general running state, a thread can further switch from being runnable to being not runnable and vice versa (there are not restrictions as to the number of times this may occur in a thread's life time). A thread may become not runnable, by:

a) Sleeping — this term refers to the case of voluntarily relinquishing control of the CPU for a given temporal interval. In other words, in this case the thread is explicitly telling the system that it will not require any time slices for a specific period of time.

b) Waiting — this term refers to the case whereby the thread has requested access to a resource that is currently not available and is waiting to be notified of the availability of that resource (at which time it can resume its activities).

c) I/O blocking — this term refers to the case of the thread waiting until some input or output activity is completed. The difference between waiting and blocking is that, in the first case, the resources are mainly programmatic entities and the notification about their availability most typically comes from other threads; in the second case, on the other hand, there is no other programmatic entity necessarily involved - the thread simply has to wait until some I/O operation is completed (e.g., until a buffer is physically written to disk).

In addition to transiting from being runnable to being not runnable as described above, threads also have the option of yielding. This term signifies the fact that the thread is allowing the CPU to run some other thread at a particular point in time, but without going to sleep. The significance of this is that, if there are no other threads that should/could be run at that point in time, control is immediately returned to the yielding thread.

Finally, a thread is said to be dead when it exits its running state and returns to the system the resources it had been using for its internal operation. Once a thread has died, it cannot return to any of the previous life cycle states. It is important to note that, although there are ways of forcefully stopping a thread "from the outside", this is not always a good idea: the thread might be in the middle of a critical task, might have hold of system resources (e.g., socket connections) that need to be explicitly released, and so on. Forcefully terminating a thread means that it does not get a chance to "clean up" and properly complete or abort its on-going activities.

As a result of the above, to protect the integrity of threads, the Java platform makes it extremely difficult to kill a running thread "from the outside". Instead, it provides facilities for implementing the recommended approach, which is to notify the thread that it should stop and then wait until the thread is finished with whatever it was doing and exits the running state by itself.

An exception to the above is daemon threads: this is a special category of threads which the system is allowed to terminate abruptly (e.g., when main program exits). In Java it requires to declare that a thread is a daemon one explicitly - usually at creation time.

Most applications require threads to communicate and synchronize their behavior to one another. The simplest way to accomplish this task in a Java program is with locks. Another way is to use a counting semaphore. These two ways are described subsequently in detail.

3.1.3 Locks

To prevent multiple accesses, threads can acquire and release a lock before using resources. Locks around shared variables allow Java threads to quickly and easily communicate and synchronize [12]. A thread that holds a lock on an object knows that no other thread will access that object. Even if the thread with the lock is preempted, another thread cannot acquire the lock until the original thread wakes up, finishes its work, and releases the lock. Threads that attempt to acquire a lock in use go to sleep until the thread holding the lock releases it. After the lock is freed, the sleeping thread moves to the ready-to-run queue.

In Java programming, each object has a lock; a thread can acquire the lock for an object by using the synchronized keyword. Methods, or synchronized blocks of code, can only be executed by one thread at a time for a given instantiation of a class, because that code requires obtaining the object's lock before execution.

Unfortunately, the use of locks brings with it many problems. They are discussed in the following, as well as their solutions.

3.1.3.1 Deadlocking

Deadlocking is a classic multithreading problem in which all work is incomplete because different threads are waiting for locks that will never be released. Although difficult to detect and hash out in every case, by following these few rules, a system's design can be free of deadlocking scenarios:

1. Have multiple threads acquire a group of locks in the same order. This approach eliminates problems where the owner of X is waiting for the owner of Y, who is waiting for X.
2. Group multiple locks together under one lock.
3. Label resources with variables are readable without blocking.
4. Most importantly, design the entire system thoroughly before writing code. Multithreading is difficult, and a thorough design before designers start to code will help avoid difficult-to-detect locking problems.

3.1.3.2 Inaccessible Threads

Occasionally threads have to be blocked on conditions other than object locks. I/O is the best example of this problem in Java programming. When threads are blocked by an I/O call inside an object, that object must still be accessible to other threads. That object is often responsible for canceling the blocking I/O operation. Threads that make blocking calls in a synchronized method often make such tasks impossible. If the other methods of the object are also synchronized, that object is essentially frozen while the thread is blocked. Other threads will be unable to message the object (for example, to cancel the I/O operation) because they cannot acquire the object lock. Be sure not to synchronize code that makes blocking calls, or make sure that a non-synchronized method exists on an object with synchronized blocking code. Although this technique requires some care to ensure that the resulting code is still thread safe, it allows objects to be responsive to other threads when a thread holding its locks is blocked.

3.1.4 Semaphores

Frequently, several threads will need to access a smaller number of resources. One way to control access to a pool of resources (rather than just with a simple one-thread lock) is to use what is known as a counting semaphore. A counting semaphore encapsulates managing the pool of available resources. Implemented on top of simple locks, a semaphore is a thread-safe counter initialized to the number of resources available for use. For example, we would initialize a semaphore to the number of database connections available. As each thread acquires the semaphore, the number of available connections is decreased by one. Upon consumption of the resource, the semaphore is released, increasing the counter. Threads that attempt to acquire a semaphore when all the resources managed by the semaphore are in use will be simply blocked until a resource is free.

A common use of semaphores is in solving the "consumer-producer problem." This problem occurs when one thread is completing work that another thread will use. The consuming thread can only obtain more data after the producing thread finishes generating it. To use a semaphore in this manner, designers create a semaphore with the initial value of zero and have the consuming thread block on the semaphore. For each unit of work completed, the producing thread signals (releases) the semaphore. Each time a consumer consumes a unit of data and needs another, it attempts to acquire the semaphore again, resulting in the value of the semaphore always being the number of units of completed work ready for consumption. This approach is more efficient than having a consuming thread wake up, check for completed work, and sleep if nothing is available.

3.1.5 Designing for Different Threading Models

The determination of whether the threading model is preemptive or cooperative is up to the implementors of the virtual machine and it varies across different implementations. As a result, Java developers must write programs that work under both models.

Under the preemptive model, threads can be interrupted in the middle of any section of code, except for an atomic block of code. Atomic sections are code segments that once started will be finished by the current thread before it is swapped out. In Java programming, assignment to variables smaller than 32 bits is an atomic operation, which excludes variables of types double and long (both are 64 bits). As a result, atomic operations do not need synchronization. The use of locks to properly synchronize access to shared resources is sufficient to ensure that a multithreaded program works properly with a preemptive virtual machine.

For cooperative threads, it is up to the programmer to ensure that threads give up the processor routinely so that they do not deprive other threads of execution time. One way to do this is to call yield method, which moves the current thread off the processor and onto the ready queue. A second approach is to call sleep method, which makes the thread give up the processor and does not allow it to run until the amount of time specified in the argument to sleep has passed.

But simply placing these calls at arbitrary points in code doesn't always work. If a thread is holding a lock (because it's in a synchronized method or block of code), it does not release the lock when it calls yield method. This means that other threads

waiting for the same lock will not get to run, even though the running thread has yielded to them. To alleviate this problem, call yield method when not in a synchronized method. Surround the code to be synchronized in a synchronized block within a non-synchronized method and call yield method outside of that block.

Another solution is to call wait method, which makes the processor give up the lock belonging to the object it is currently in. This approach works fine if the object is synchronized at the method level, because it is only using that one lock. In addition, a thread that is blocked on a call to wait method will not awaken until another thread calls notify method, which moves the waiting thread to the ready queue. To wake up all threads that are blocking on a wait method call, a thread calls notifyAll method.

3.1.6 Threads and AWT/Swing

In order to provide a friendly interface for users and make EM simulation convenient, Java GUI should be developed for the web-based EM simulation system with Java Swing or AWT.

In Java programs with GUIs that use Swing and/or the AWT, the AWT event handler runs in its own thread. Developers must be careful to not tie up this GUI thread performing time-consuming work, because it is responsible for handling user events and redrawing the GUI. In other words, the program will appear frozen whenever the GUI thread is busy. Swing callbacks, such as Mouse Listeners and Action Listeners are all notified (by having appropriate methods called) by the Swing thread. This

approach means that any substantial amount of work the listeners are to do must be performed by having the listener callback method spawn another thread to do the work. The goal is to get the listener callback to return quickly, allowing the Swing thread to respond to other events. If a Swing thread is running asynchronously, responding to events and repainting the display, Swing callbacks can safely modify Swing data and draw to the screen.

Having a non-Swing thread modify Swing data is not thread safe. Swing provides two methods to solve this problem: invokeLater() and invokeAndWait(). To modify the Swing state, simply call either of these methods, passing a Runnable object that does the proper work. Even though Runnable objects are usually their own threads, they would not be thread safe. Instead, Swing puts this object in a queue and executes its run method at an arbitrary point in the future. This makes the changes to Swing state thread safe.

According to the techniques which are described in Chapter 2 and this chapter, two examples will be given later to illustrate the development of EM simulation system on the web.

3.2 Development of Wed-based HFSS System

This section describes the development of the web-based system by porting the EM Simulator HFSS.

3.2.1 Ansoft HFSS

The High Frequency Structure Simulator (HFSS) is a high frequency electromagnetic simulation software, developed by Ansoft. It employed Finite Element Method as the solver. The simulator has been widely used for electromagnetic modeling and simulation ranging from simple wire antenna to fairly complex microwave and electronic devices.

3.2.1.1 Features of Ansoft HFSS

Two of the most distinct features of Ansoft HFSS [16] are its ability to model the resonance and the quality factor of microwave cavities as well as a more broadband fast sweep capability. Another feature is its ability to model active devices and phased-array antennas through fields linked by values at two different boundaries. Its higher accuracy mesh truncation procedures for modeling radiation from antennas, more powerful geometry modeling using precise solids and faster, more complete graphics tools for post-processing are also additional features used widely by many users.

Ansoft HFSS is an interactive software package for calculating the electro-magnetic behavior of a structure. The simulator also includes post-processing commands for analyzing the electromagnetic behavior of a structure in more detail. Ansoft HFSS has the following capabilities:

1. Computing electromagnetic field quantities for open boundary and near-field problems.

2. Calculating characteristic port impedances and propagation constants.
3. Calculating generalized S-parameters and S-parameters renormalized to a specific port impedance.
4. Estimating eigenmodes or resonances of a structure.

Designers can build the problem model, specify material characteristics for each object, and identify ports, sources, or special surface characteristics. The system then generates the necessary field solutions. After designers set up the problem, Ansoft HFSS allows them to specify whether to solve the problem at one specific frequency or at several frequencies within a range.

3.2.2 Web Enabling

Ansoft HFSS is available on UNIX workstations running at X Windows, Windows NT, and Windows 2000. However, modeling and simulation of a complex electromagnetic problem often needs a high capacity and expensive supercomputer. The investment is not easily available to many small and medium sized companies. Therefore, there is a strong motivation to develop the HFSS Simulator with portability to offer an easy and convenient approach with which designers can perform electromagnetic simulation remotely.

The rapid advancement in Internet accessibility and wide bandwidth technology allow the motivation turn into realization. In order to make the HFSS package available on the web for engineers and designers, it is necessary to enable HFSS simulation at different platform on the World Wide Web. In the development of web-based HFSS system, a middleware—Tarantella (TTA) is used.

3.2.3 A Middleware—Tarantella

Tarantella (TTA) Enterprise software provides a non-intrusive solution and fast, secure access to different platforms, including Microsoft Windows, web-based, Java™, mainframe, AS/400, Linux, and UNIX systems, and applications from client devices anywhere [17]. This proven solution centralizes management, reduces complexity, and scales to accommodate rapid corporate change, technological advancement, and expanding remote access needs.

With Tarantella Enterprise software, users can access applications remotely from their client devices (anything from a thin client to a top-of-the-range PC). All that users need is a web browser (such as Microsoft Internet Explorer or Netscape Navigator), with Java technology enabled. The advantage of this approach:

- Eliminates the need to install additional software on the client device,
- Dramatically reduces the time to deliver applications,
- Extends the reach of the applications,
- Increases manageability, and
- Centralizes management of users and applications

The above sections discussed the functions and features of Ansoft HFSS and TTA. Now the question is how the system works. To answer this question, a three-tier architecture model is implemented. The following section describes the development and structure of this three-tier architecture.

3.2.4 The Three-Tier Architecture

The web-based HFSS simulation system is built using a three-tier architecture model. It includes the client devices tier, middle tier and application servers tier, as shown in Fig 3.4.

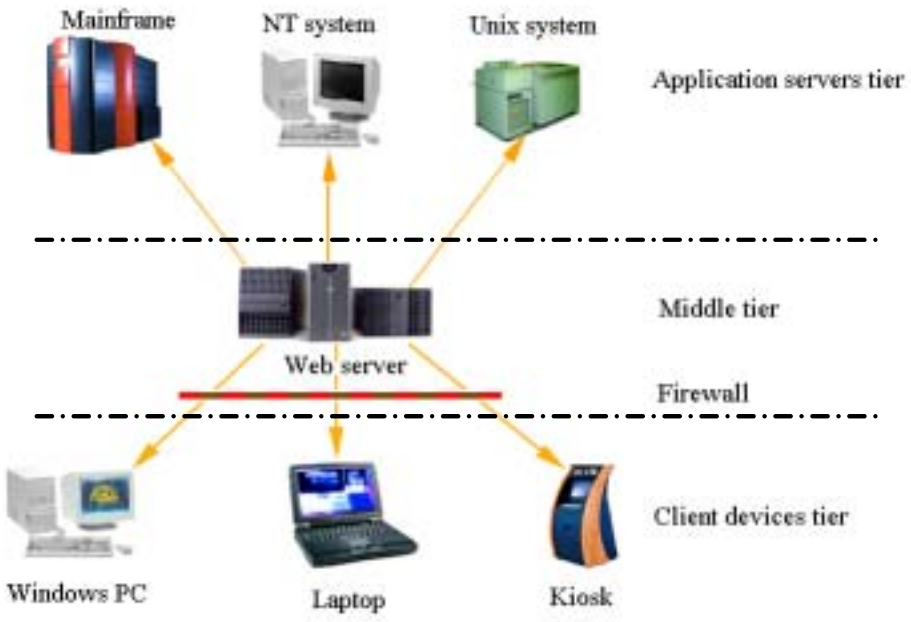

Figure 3.4: The three-tier architecture model

3.2.4.1 The Front Tier

The first tier, that is client devices tier, includes the client devices. A client device can be a Windows PC, laptop or kiosk. It is a piece of hardware that can communicate with the middle tier using a web browser. They are thin clients and there is no special application installation and configuration for running the applications hosted on the application server tier. The only requirement is a Java enabled web browser.

3.2.4.2 The Second Tier

The second tier, that is middle tier, includes the broker hosts. The broker host is a piece of hardware running a web server and an application broker. The web server generates documents that are sent to client devices. Java applets embedded in these documents allow direct communication between client devices and the application broker server. The applets use specific protocols, such as TCP/IP, X protocol or even Adaptive Internet Protocol (AIP), which is a protocol, adapts to the client devices being used and the applications being delivered, so that to provide optimum performance. The web server also has the responsibilities of negotiating with application servers to authenticate the users and keeping track of running applications even after users have logged out, so that they can be resumed later.

The applications broker responses to the communication between the application servers and the web server. It generates a compressed data stream going through the application servers and client devices by using a dedicated protocol. The compressed data stream is finally interpreted through the Java applets downloaded to the client web browser. In this way, the application's users interface is transferred from the application server to the clients. Users do their work in the transferred interface in their local clients.

3.2.4.3 The Third Tier

The application servers compose the third tier, where Ansoft HFSS is installed. When a user clicks a link, the request is sent to the web server via the browser. Then the web server passes the request to the application broker and the application broker starts the

application on an appropriate application server. The data generated from the application server is compressed and redirected by the application broker from the application server to the client device. This actually is the key mechanism in the system regarding how to launch an application and display it remotely through Internet on the client devices.

The web server used in the system is Apache web server. It is an open source application. As for the application broker, there are products like Tarantella® and Citrix® in the market. The application broker used in the web-based HFSS simulation system is Tarantella® server.

The three-tier is the key element of the system architecture for applications launching via Internet. This architecture is enough for accessing the hosted applications remotely.

3.2.5 Implementation of Web-enabling HFSS System

To make HFSS available on the web and develop the web-based HFSS simulation, the middleware—Tarantella Enterprise software [17] and Java relative technology will be used. As described above, Tarantella Enterprise software acts as middleware to store information on designers and applications centrally. To access the applications, designers are authenticated to the Tarantella Enterprise server. The server will automatically check its datastore of the designers and the application information to determine which applications each designer is allowed to access. This set of applications is then presented to them in the form of a 'webtop'. At all times,

Tarantella Enterprise software manages the connections, user sessions, and security. The flowchart of this system is shown in Fig 3.5.

Tarantella Enterprise software is designed as a modular, scalable, and flexible solution. It is ideally suited for use by data centers of all types, including Application Service Providers (ASPs) and other service providers that need to deploy a mixture of applications to large numbers of users, with a range of client devices, and varying connectivity types.

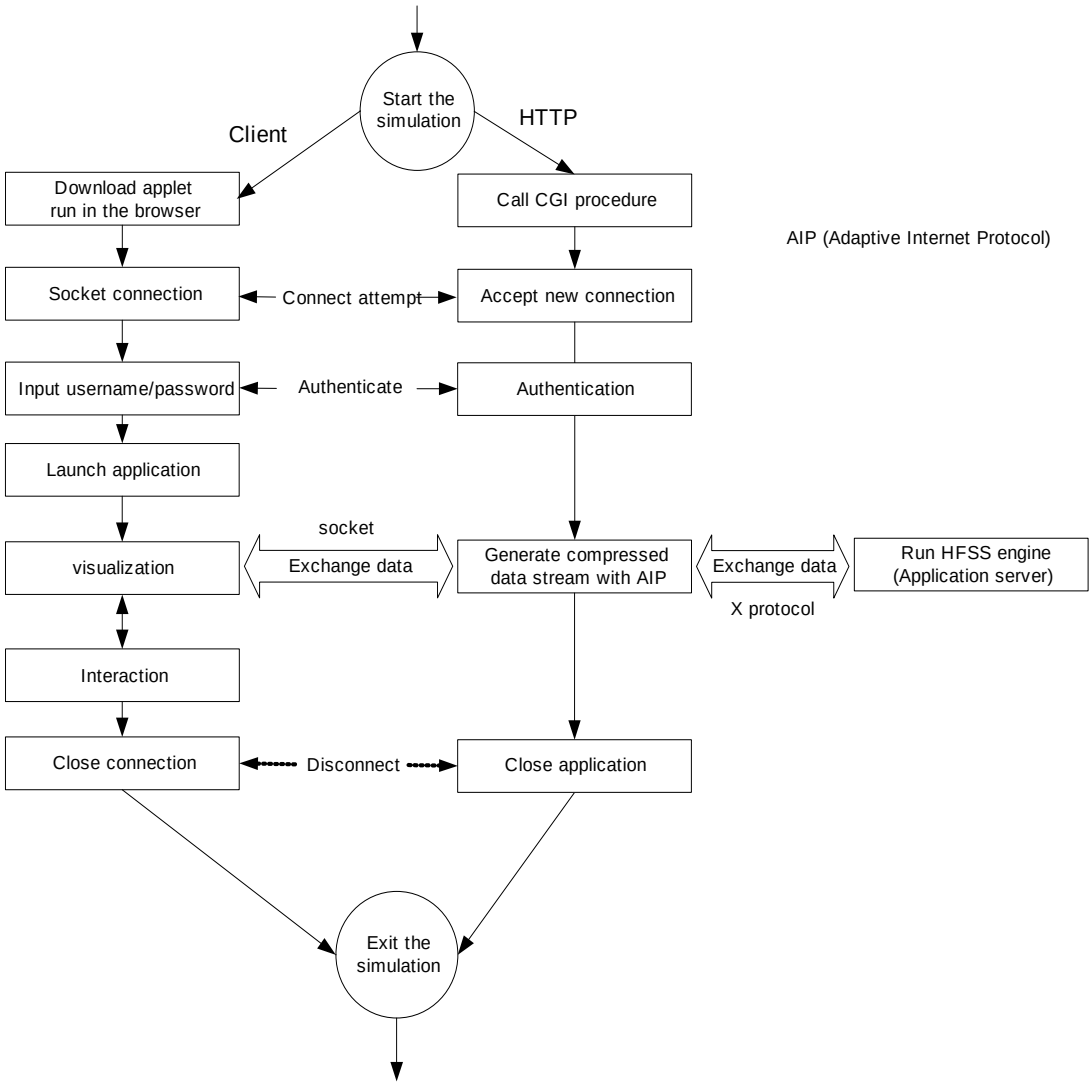

Figure 3.5: The flowchart of web-based HFSS system

■ An example of simulating dual-mode horn

After inputting the URL address to the website of simulation system, the web-based HFSS system starts to simulate. After the Ansoft HFSS is started, the interface surface is shown in Fig3.6. Designers can create a new project or open an existed project to do simulations just like doing it locally.

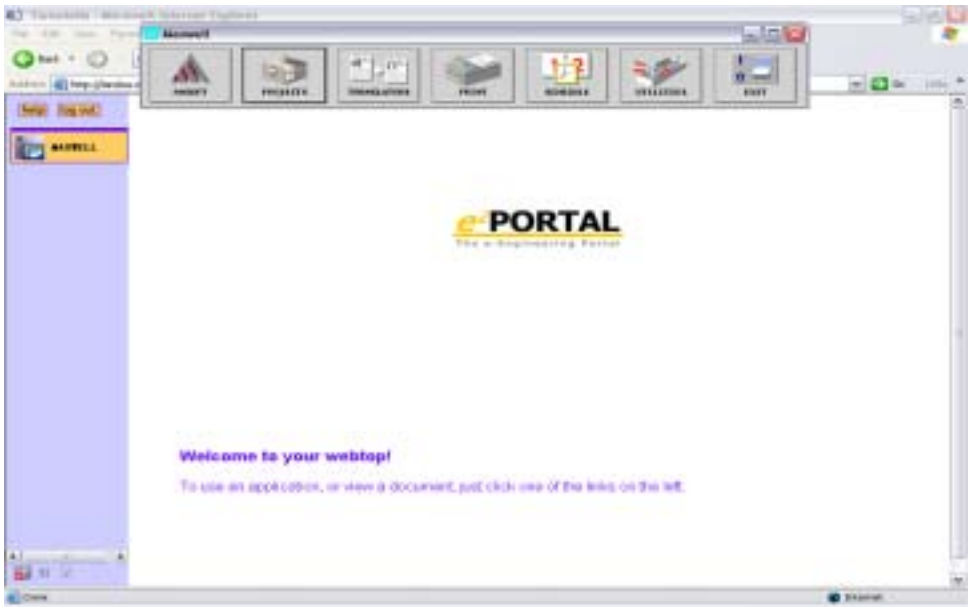

Figure 3.6: The interface of system log-in

The dual-mode horn is composed of two cylinders separated by a conic section. The conic section is where the radius of the horn increases from the first cylinder to the second. At the step where the radius changes, there is a partial conversion of TE_{11} mode energy to TM_{11} mode energy. This problem is described and analyzed in details [18]. The following section describes how to implement web-based HFSS simulation through the setup, solution and analysis of a dual-mode horn. The steps are explained in details as follows:

(1) Finite element method and antenna problem setup

In Ansoft HFSS, once the geometric model is built up, the pre-processor will automatically mesh it into a large number of elements. The collection of the elements is referred to as the finite element method. Dividing a structure into thousands of smaller regions (elements) allows the system to compute the field solution separately in each element.

Because of the symmetry in the problem, only half of the geometry is to be created. The following simulated results will be examined:

- Calculate the VSWR (voltage standing wave ratio) for the horn;
- Plot the far-field antenna pattern;
- Create an animated cloud plot of the magnitude of the electric field.

(2) Creating the antenna project and Drawing the geometric model.

At this stage, the project is created and the solver is used to obtain a solution. The geometry for the antenna problem consists of two objects: the horn antenna and the virtual object for the radiation boundary.

The solution is to use the finite element based solver to obtain a solution for a structure that is “driven” by a source. Ansoft HFSS calculates the S-parameters of passive, high-frequency structures. The simulator also includes post-processing commands for analyzing the electromagnetic behavior of a structure in more details.

The solution provides:

- Basic electromagnetic field quantities, for open boundary problems, radiated near and far fields.
- Characteristic port impedances and propagation constants, and
- Generalized S-parameters and S-parameters renormalized to specific port impedances.

The geometric model built up at HFSS through the web is shown in Fig 3.7.

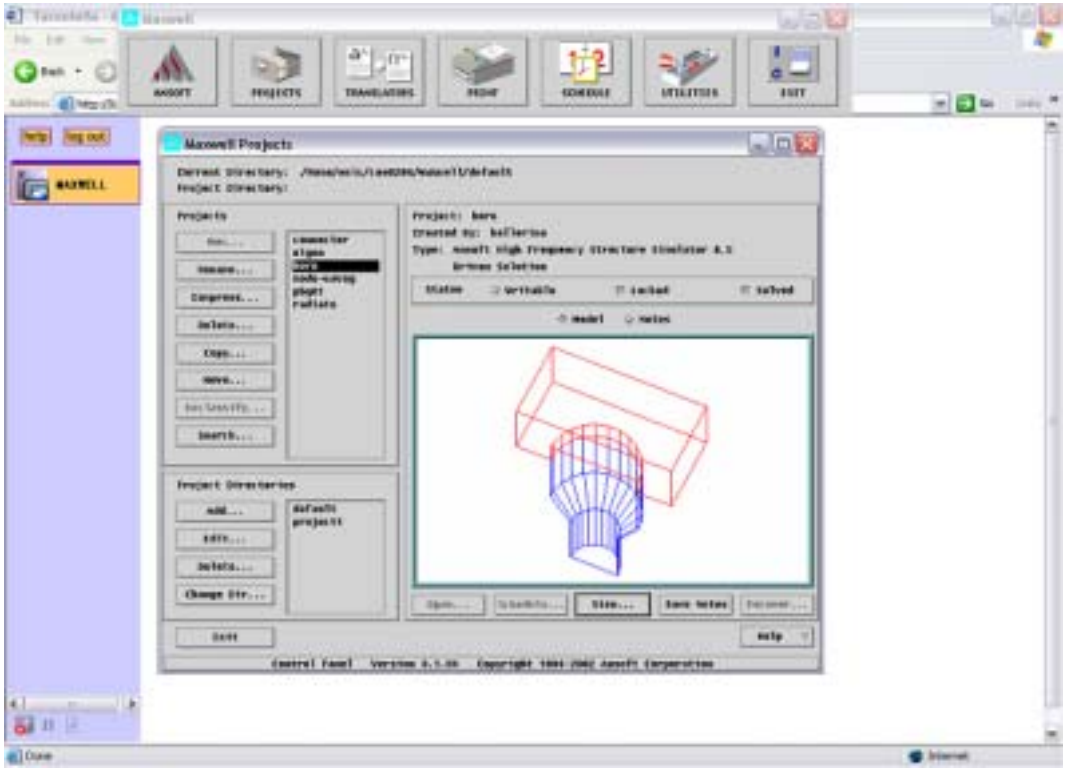

Figure 3.7: The geometric model of dual-mode horn antenna

(3) Setting up the problem

Goals of this step are to assign the material properties to all model objects in the geometric model, identify the port through which the wave enters the horn, and define the boundary conditions.

Vacuum is assigned to both the horn and box. A perfect *E* boundary will be used to simulate a perfect conductor on the surface of the horn and a radiation boundary to simulate an absorbing boundary on the box. One port is needed and four types of boundary conditions, including radiation, perfect *E*, symmetry and perfect *H*, will be used in this problem.

(4) Generating a solution

The simulator computes the 3D electromagnetic field inside the structure with excitation signals at each port. Once the field solution is obtained, the simulator computes the generalized S-matrix associated with the structure.

(5) Analyzing the solution

In this step, the goals are to plot the far field and create an animated cloud plot of the magnitude of *E* field.

► Creating the far-field plot

When calculating radiation fields, the values of the fields over the radiation surface are used to compute the fields in the space surrounding the device. This space is typically split into two regions—the near-field region and the far-field region. In general, the electric field *E(x, y, z)* external to the region bounded by a closed surface may be written as:

$$E(x,y,z) = \int_s (< j\omega\mu_0 H_{tan} > G + < E_{tan} \times \nabla G > + < E_{normal} \nabla G >) ds$$

where:

- *s* represents the radiation surfaces;
- *j* is the imaginary unit, $\sqrt{-1}$;
- ω is the angular frequency, $2\pi f$;
- μ_0 is the relative permeability of the free space;
- H_{tan} is the component of the magnetic field that is tangential to the surface;
- E_{normal} is the component of the electric field that is normal to the surface;
- E_{tan} is the component of the electric field that is tangential to the surface;
- *G* is the free space Green's function, given by:

$$G=\frac{e^{-jk_0|r-r'|}}{|r-r'|}.$$

Where:

■ k_0 is the free space wave number, $\omega\sqrt{\mu_0\varepsilon_0}$.

■ r and r' represent field points and source points on the surface respectively.

In the far field where $r >> r'$ (and usually $r >> \lambda_0$), Green's function can be approximated as: $G \approx \frac{e^{-jk_0 r}e^{jk_0\hat{r}\cdot r'}}{r}$ (here, r̂ is radial unit vector). When this form of G is used in the far-field calculations, the fields have an *r* dependence in the form of $\frac{e^{-jk_0 r}}{r}$. This r dependence is characteristic of a spherical wave, which is a key feature of far fields. The far field is a spherical TEM wave with the following equation: $E = \eta_0 H \times \hat{r}$, where η_0 is the intrinsic impedance of free space.

► Computing the far field

In this step, the previously discussed far-field approximations are used. The result is valid only for field points in the far-field region.

► Plotting the far fields

After the far fields are computed, the plotting parameters of the far field are specified and the plot is displayed in this step. The two-dimensional E-total field (on the theta plane where phi is set to a fixed variable) is shown in Fig 3.8. The maximum occurs at the zero degree in the theta plane.

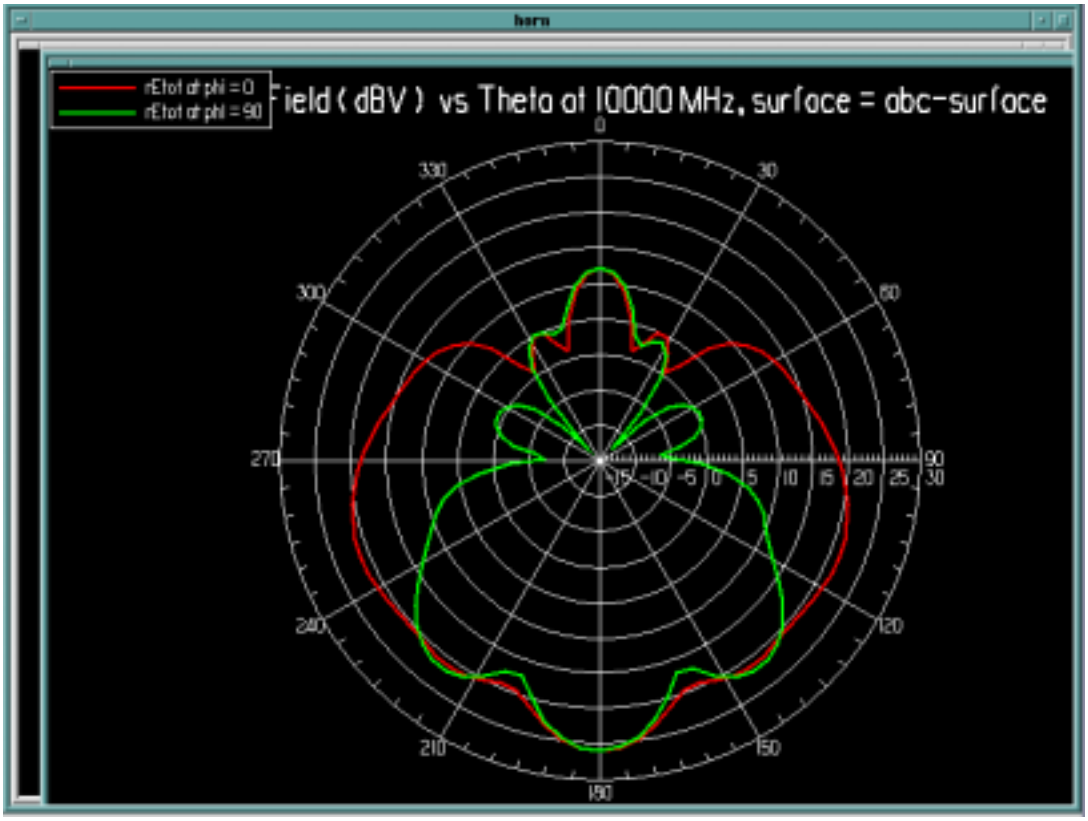

Figure 3.8: The plot of E total fields

► Creating an animated clout plot

An animated plot is created frame by frame. In this step, a series of pictures of the magnitude of the E-field are generated, while varying its phase. Each picture represents the E-field at a different phase. One of these frames is shown in Fig 3.9.

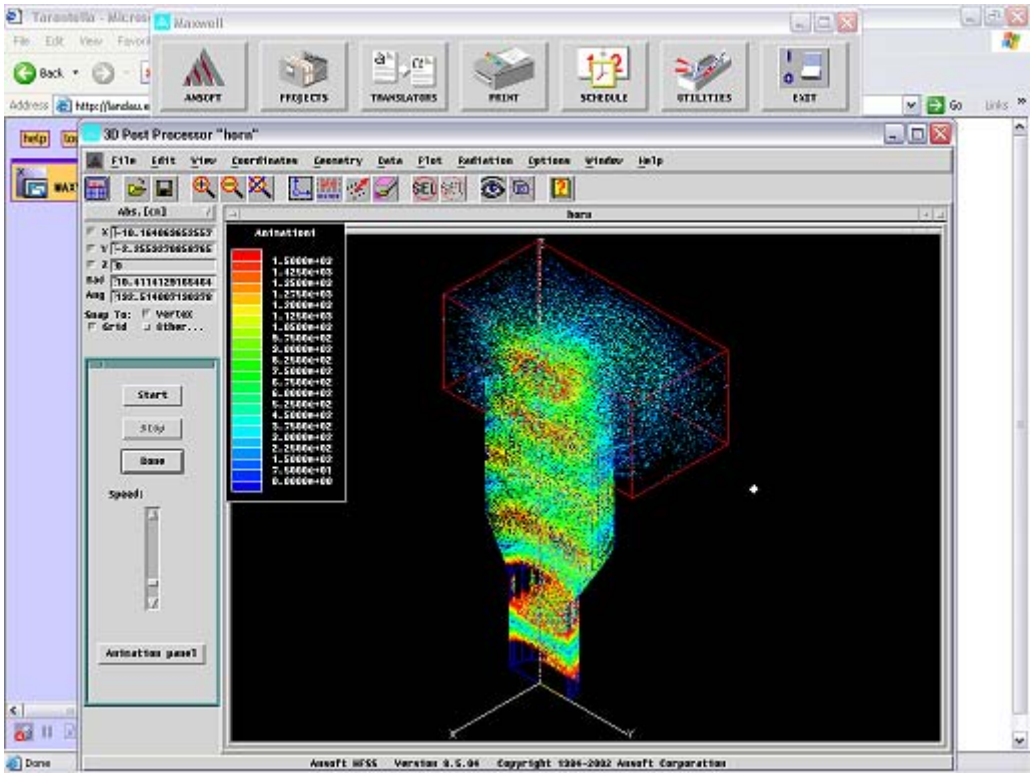

Figure 3.9: An animated clout plot

3.3 Conclusions

Java's multithreading capability provides the means to build web-based applications with many concurrent threads of activity, which results in a high degree of user interactivity. Multithreading technique enables the server to be capable of serving multiple clients, and thus make electromagnetic simulation accessible simultaneously on line.

In this chapter, with Java techniques the development of web-based HFSS system solves the performance and maintenance requirement for computer, and make designers implement HFSS simulation remotely and simultaneously. Simulation and analysis essential to the engineering community are provided on the system.

Chapter 4 Web-based FDTD Simulation System

Another example of web-based EM simulation system, which is the web-based FDTD simulation system, is presented in this chapter. In this system, Servlets and Applet, and Java socket techniques are described in details. The communication between applet and servlet and socket communication between web server and FDTD package are solved and implemented. In the latter part of this chapter, the implementation of a simulation is given, and its result is presented.

4.1 Introduction

The universe of web-based applications includes architectures that emphasize client-side applications, “applets”, architectures that emphasize server-side applications, “servlets”, and their combinations. The combination of applets and servlets provides a wide spectrum of choices in deciding where and what computation is performed in a web application. One of the concerns in building servlets is the performance and security that, ultimately, depends on the language used to implement the server-side computation. Common Gateway Interface (CGI) scripts were first approach to server-side dynamic content generation. Proprietary server APIs developed as a result but the need for a flexible programming language with security features led to a servlet API between server and servlets.

Java servlets provide a new and exciting method of developing server-side solutions [19]. Servlets provide the features of traditional CGI scripts with the added benefits of efficiency and portability. Currently, major corporations are making the migration from CGI scripts to Java servlets.

Therefore, in the web-based FDTD simulation system, servlets are used to develop server-side applications, and applets are used to develop client-side. As a result, the demand for applet and servlet communication is on the rise.

4.2 Ways of Communication between Applet and Servlet

There are four techniques that allow applets and servlets to communicate with each other: HTTP streams, using the java.net packages to create a direct network connection, invoking a remote method using Java's remote method invocation (RMI) interface, and, finally, using CORBA. This part will review each of these four techniques.

4.2.1 HTML Streams

4.2.1.1 HTTP Text Streams

The simplest way for an applet to exchange information with a servlet is through an HTTP text stream [10]. Java's URL and URLConnection classes make it easy to read data from a URL without having to worry about sockets and other normally complex issues of network programming.

However, Using simple text streams to exchange data has one major weakness that the applet needs not only to know the format of the data but also to perform parsing and conversion of the data into a useful form.

In the following simple doGet method, the servlet writes an HTML page embedding an applet called "simpleApplet" that takes a parameter called Data.

The doGet method is defined with return type and parameters, and GET requests are for getting limited amounts of information appended to the URL. The parameter list for the doGet method takes a request and a response object. The browser sends a request to the servlet and then the servlet sends a response back to the browser. The doGet method implementation accesses information in the request object to find out who made the request, what type of form the request data is in, and which HTTP headers were sent, and uses the response object to create an HTML page in response to the browser's request. The doGet method throws an IOException if there is an input or output problem when it handles the request, and a ServletException if the request could not be handled. These exceptions are handled in the HttpServlet class.

```
public void doGet (HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
response.setContentType ("text/html");
  PrintWriter out = new PrintWriter (response.getOutputStream ());
  out.write ("<HTML><HEAD><TITLE> HTML Manipulation Example</TITLE>
      </HEAD><BODY>");
  out.write ("This applet receives data from the server via its PARAM tag");
  out.write ("<applet code=\"SimpleApplet\" width=\"225\" height=\"149\">");
  //Manipulate param tag
  out.write ("<param name=\"Data\" value=\">");
  out.write (java.text.DateFormat.getDateInstance ().Format (new java.util.Date ()););
  out.write ("\">"</APPLET></BODY></HTML>);
  out.close ();
  }
```

4.2.1.2 HTTP Object Streams

An HTTP connection can be used to transfer binary data as well as textual data. We can combine this capability with object serialization, which is discussed in Chapter 2, to pass complete Java objects from a servlet to an applet. Complex data can be transferred very easily through this way with no need for parsing and interpretation.

Object serialization allows designers to "flatten" objects into streams of binary data that can go anywhere an OutputStream can go, across an HTTP connection to an applet.

Use of an HTTP object stream is in very much the same way as to use an HTTP text stream [10]. Programmers initiate a URL connection back to a servlet on the web server, then read in the resulting data. Rather than wrapping the InputStream in a DataInputStream as before, we wrap it in an ObjectInputStream—a special type of stream that reads objects. We then read in each object, casting it to the appropriate type.

4.2.2 Two-Way Talk with java.net

The second technique is to use the java.net package's networking capabilities [20]. In this situation the applet does not speak directly with the servlet. Rather, the servlet creates a listener class during initialization. This listener class creates a ServerSocket and listens for incoming connections from the applet. When an incoming connection is received, the listener class hands off the data connection to a third class that communicates with the servlet using Java's standard I/O library.

The java.net technique is quite clean and fairly easy to implement. Because it relies on nothing more than Java's standard network interfaces, it's easy to modify for a particular need. On the other hand, you must parse the various requests for information coming in at the server side and then, on the client side, interpret the results coming back. If there are many server-side methods to use, RMI can be considered.

4.2.3 Remote Method Invocation (RMI)

Java's RMI (Remote Method Invocation) technology significantly increases programmer's ability to work with complex server-side objects [19]-[20]. To the applet, a server-side object looks like a regular client-side handle. This technique relies on object-oriented polymorphism. The RMI API allows an applet to invoke the methods of a Java object executing on the server machine, and, in some cases, it also allows the object on the server machine to invoke the methods of the applet.

In fact, RMI also uses object serialization to pass objects back and forth between the client application and the remote object. All of the low-level details of network connections and serialization are hidden from the developer when using RMI.

The advantages of RMI for applet-server communication are as follows:

- It allows applets and server objects to communicate using an elegant high-level, object-oriented paradigm;
- It allows server objects to make callbacks to the methods of the applet; and
- It can be made to work through firewalls.

The disadvantages of RMI are equally concerning:

- It's complicated;
- It's supported in few browsers; and
- It can be used only by Java clients.

RMI is appropriate for situations where you have either dynamic or large data provided by Java objects on the server. Although RMI does allow for OO distributed programming, it is only supported when both client and server are written in Java. If the application demands mixed language development, RMI will not be sufficient and CORBA must be used. The following code segment is an example of the design with RMI. After we define the serv2app public interface, we must write a class that extends UnicastRemoteObject and implements the interface. This class must have a default constructor that is defined as throwing RemoteException.

Running RMI is appropriate when dealing with complex server-side objects such as dynamic data or large amounts of data. The process involves three steps: starting the registry, running RMIImplementation, and starting your Web server. The RMIImplementation class extends UnicastRemoteObject and implements user interface. In addition, this class must have a default (no argument) constructor that is defined as throwing RemoteException shown as follows.

```
import java.rmi.*;
import java.rmi.server.*;
public class RMIImplementation extends UnicastRemoteObject
  implements RMIServ2App{
  public RMIImplementation() throws RemoteException{}
```

```
    public String getData() throws RemoteException{
    return java.text.DateFormat.getDateInstance().format(
      new java.util.Date());
    }
  public static void main(String[] args){
    System.setSecurityManager (new RMISecurityManager ());
    try{
      RMIImplementation bootStrap =
        new RMIImplementation();
      Naming.bind (args [0], bootStrap);
      System.out.println ("Inserted RMIImplementation
        into registry at " + args[0]);
      System.out.println ("Verify this name in your
        servlet.properties file.");
    } catch (Exception e) {
      System.err.println ("Failed to insert bootstrap
        object into registry");
      System.err.println (e);
    }
  }
}
```

4.2.4 Common Object Request Broker Architecture (CORBA)

Like RMI, CORBA (Common Object Request Broker Architecture) allows programmers to make method calls on remote objects [20]. If there is legacy server-side code written in a different language, then we can wrap it as a CORBA object and expose its functionality. CORBA provides a rich framework of services and facilities for distributing objects on the network.

In terms of programming, CORBA works much like RMI: to define an interface, use a tool to create stubs and skeletons, implement the interface, and register it with a server-side service which, to the client, looks just like a local object. The major differences are that CORBA uses a language-neutral interface description language (IDL) to specify the interface (allowing for mixed-language development) and CORBA is a much more extensive set of protocols. The big disadvantage of CORBA is that the client must be able to call the ORB interfaces locally. This generally involves the distribution of a hefty JAR file, although some ORBs provide relatively lightweight distributions. (ORB vendors don't appear to compete on distribution size. Performance, reliability, and software tools seem to be the key issues.). The following example is to make a Server-Side Class Available to Others. In this init () method of CORBAServlet, the ORB is initialized with a single call as is the Basic Object Adapter BOA, which is similar to the RMI registry. Programmers can create an instance of their implementation, insert it into the BOA, and tell the BOA that it is now ready to accept requests.

In this init() method of CORBAServlet, the ORB is initialized with a single call as is the Basic Object Adapter BOA, which is similar to the RMI registry. Designers can create an instance of their implementation, insert it into the BOA, and tell the BOA that it is now ready to accept requests.

```
public class CORBAServlet extends HttpServlet{
  String CORBAName;
  public void init(
    ServletConfig config) throws ServletException {
    super.init (config);
    try {
      CORBAName = getInitParameter ("CORBAName");
```

```
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init ();
      org.omg.CORBA.BOA boa = orb.BOA_init ();
      com.servletsolutions.serv2app.corba.CORBAServ2App implObject =
        new CORBAImplementation (CORBAName);
      boa.obj_is_ready (implObject);
      boa.impl_is_ready ();
    }
    catch (Exception e) {
      e.printStackTrace ();
    }
  }
```

4.2.5 Comparison

As a project gets more complex, Java's approach may become more complex but also more powerful. A console application can first evolve from a GUI using a simple LayoutManager, and then become a full-featured Swing application with a great interface. Similarly, the task of client/server communication advances from HTML streams and direct manipulation of the applet's parameters, to use of the socket communication with the java.net package, to CORBA and RMI.

However, in situations when the calculation of the server-side data is large compared to network lag (such as when accessing large databases or legacy applications), or in situations where you want to expose a large set of server-side capabilities without knowing the applications that will use them, RMI and CORBA are appropriate. With the large number of ORBs available and the maturity of the CORBA specifications, CORBA is preferable in most situations. RMI and CORBA supposedly provide

network transparency. In truth, the network is never transparent, and when you're talking about the Internet, the network is pretty darn close to opaque.

HTTP object streams give designers a very convenient way to exchange complex collections of information between the applet and the server, so this way is used in the development of web-based FDTD simulation system. In this system, an applet is created and uses a POST method to send a serialized object to a servlet. The appropriate server-side code for the servlet is provided for reading in a serialized object. Applets used this communication method to send a true Java object to a servlet. The servlet was also enhanced to return a vector of result object to the applet. Likewise, the appropriate applet code is provided for reading in a vector of result object. The details of its use will be described later.

4.3 Socket Communication

A socket is one end-point of a two-way communication link between two programs running on the network [21]-[24]. To communicate over TCP, a client program and a server program establish a connection to one another. Each program binds a socket to its end of the connection. To communicate, the client and the server each reads from and writes to the socket bound to the connection.

Sockets have two major modes of operation: connection-oriented and connectionless modes. The mode used is determined by an application's needs. Based on the requirement of the web-based FDTD system, connection-oriented modes is selected.

The socket classes are used to represent the connection between a client program and a server program. The java.net package provides two classes, Socket and ServerSocket, that implement the client side of the connection and the server side of the connection, respectively [21].

4.3.1 Java Connection-Oriented Classes

4.3.1.1 Client socket

Recall that a client socket issues a connect call to a listening server socket. Client sockets are created and connected by using a constructor from the Socket class. The following line creates a client socket and connects it to a host:

```
Socket clientSocket = new Socket ("fdtdserver", 80);
```

Because the Socket class is connection-oriented, it provides a streams interface for reads and writes. Classes from the java.io package should be used to access a connected socket:

```
DataOutputStream outbound=new DataOutputStream
(clientSocket.getOutputStream ());
BufferedReader inbound = new BufferedReader (new
InputStreamReader (clientSocket.getInputStream ()));
```

Once the streams are created, normal stream operations can be performed.

4.3.1.2 Server Sockets

Servers do not actively create connections. Instead, they passively listen for a client connect request and then provide their services. Servers are created with a constructor from the ServerSocket class. ServerSocket is a java.net class that provides a system-independent implementation of the server side of a client/server socket connection. The following line creates a server socket and binds it to port 80:

```
ServerSocket serverSocket = new ServerSocket (80, 5);
```

A server can receive connection requests from many clients at the same time, but each call is processed one at a time. The listen stack is a queue of unanswered connection requests. The preceding code instructs the socket driver to maintain the last five connection requests. If the constructor omits the listen stack depth, a default value of 50 is used.

Once the socket is created and listening for connections, incoming connections are created and placed on the listen stack. The accept method is called to lift individual connections off the stack:

```
Socket clientSocket = serverSocket.accept ();
```

The accepted method waits until a client starts up and requests a connection on the host and port of this server. When a connection is requested and successfully established, the accepted method returns a new Socket object which is bound to a new port. The server can communicate with the client over this new Socket and continue

to listen for client connection requests on the ServerSocket bound to the original, predetermined port.

After the server has successfully established a connection with a client, it communicates with the client using this code:

```
PrintWriter out = new PrintWriter (clientSocket.getOutputStream (), true);
BufferedReader in = new BufferedReader (new InputStreamReader (
                    clientSocket.getInputStream ()));
String inputLine, outputLine;

while ((inputLine = in.readLine ()) != null) {
   ...
      break;
}
```

The flowchart in Fig 4.1 summarizes the steps needed for client/server connection-oriented applications.

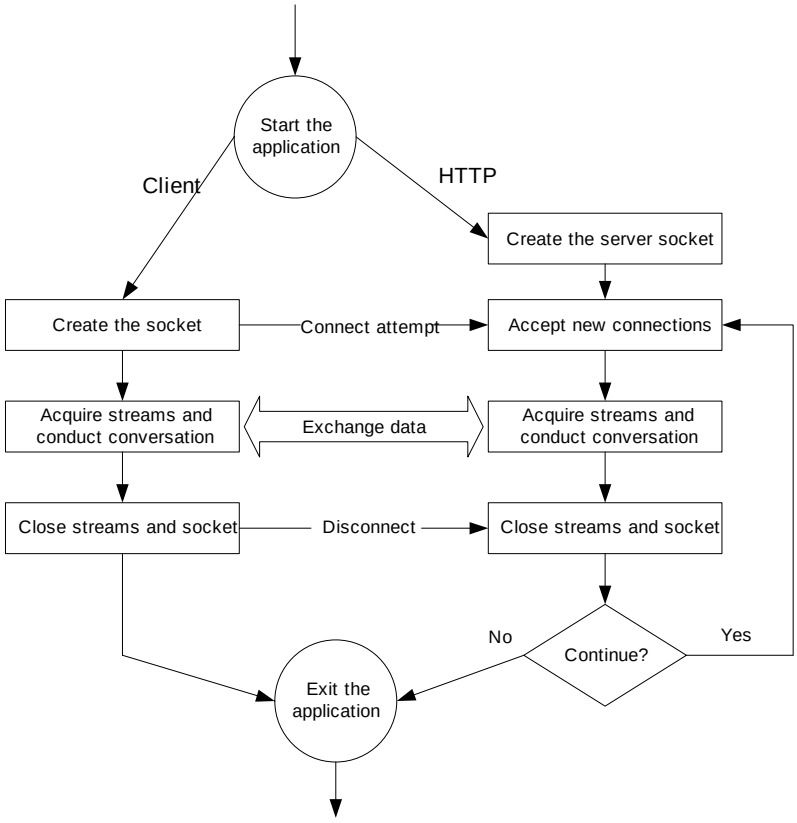

Figure 4.1: The flowchart of client/server connection-oriented applications

4.3.2 Socket and Thread

The socket associates the server program with a specific hardware port on the machine where it runs any client program anywhere in the network with a socket associated with that same port can communicate with the server program.

A server program typically provides resources to a network of client programs. Client programs send requests to the server program, and the server program responds to the request.

The way to handle requests from more than one client is to make the server program multi-threaded [22]. A multi-threaded server creates a thread for each communication

it accepts from a client. A thread is a sequence of instructions that run independently of the program and of any other threads. Using threads, a multi-threaded server program can accept a connection from a client, start a thread for that communication, and continue listening for requests from other clients.

4.4 Development of Web-based FDTD Simulation System

By developing a web-based FDTD simulation system, it is able to implement FDTD simulation and visualize its results via the Internet. The flow chart of the web-based FDTD simulation system is given in Fig 4.2. Communications between the web sever and the supercomputer where FDTD engine is installed, is shown in Fig 4.1. In Fig 4.1, FDTD engine is on the server socket side.

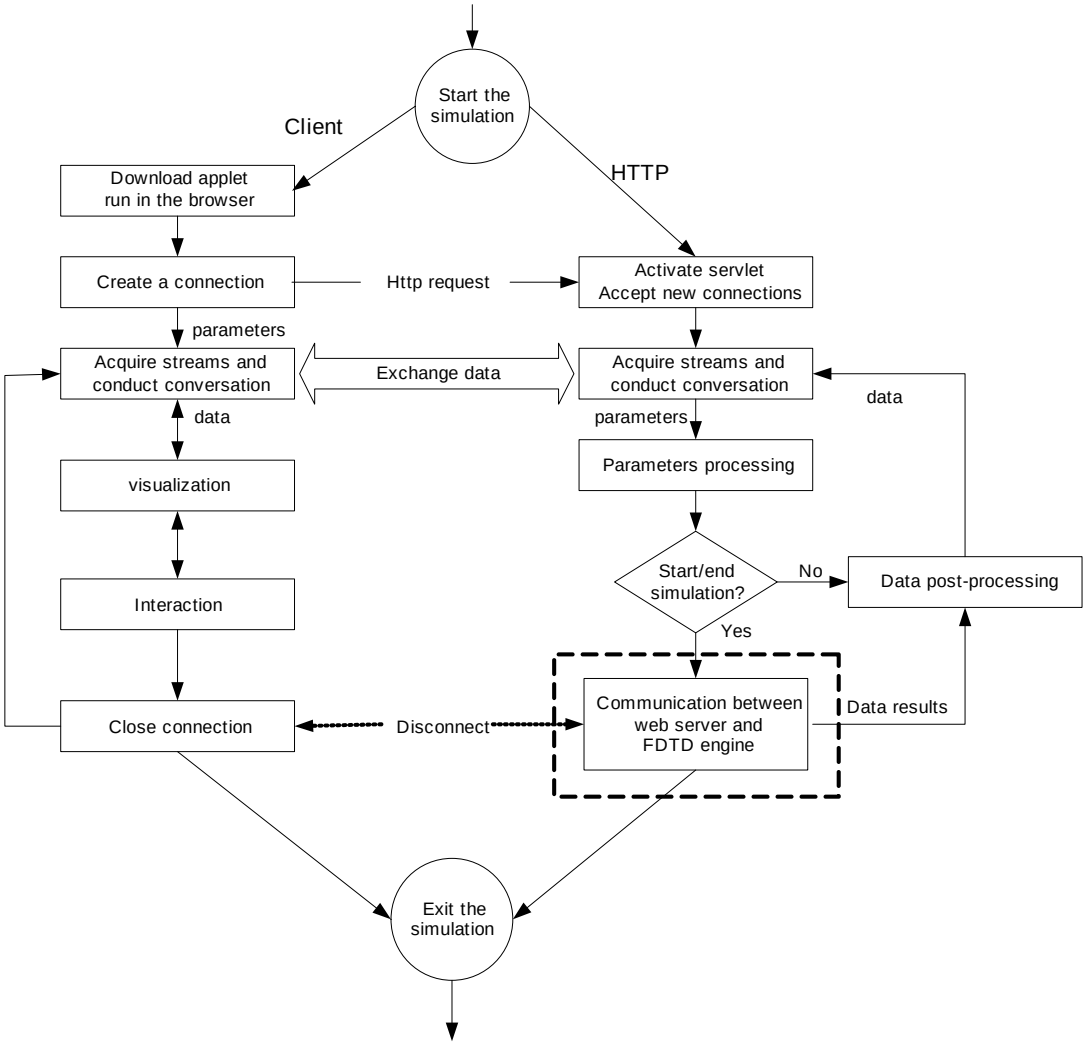

Figure 4.2: The flow chart of web-based FDTD simulation system

This section will address how to develop the FDTD simulation at web environment through GUI. GUI is to provide a graphic interface for applications and to make an application more user friendly. With GUI, the user need not take much time to remember functions of key sequence, but use the application itself. The user and application interact and communicate with each other through GUI.

Here Java will still be used to develop web-based FDTD simulation system. A feature of Java is that the "look-and-feel" of the GUI which can be chosen independently of

the underlying platform. The pluggable look and feel can design a single set of GUI components that can automatically have the look and feel of any OS platform (Microsoft Windows, Solaris^{TM}, and Macintosh). As shown in Fig 4.2, the servlet is used in the server-side and applet is used in the client-side. Sockets are used to handle the communication between the web server and the FDTD package. Detailed discussion will be made subsequently.

4.4.1 Servlet Implementation

All servlets implement the Servlet interface, which is part of the Servlet API [25]. Usually, this is undertaken by extending the HttpServlet class, and in this simulation system there is no exception. When a server loads a servlet, it runs the servlet's initialization method, init.

After the server has loaded and initialized the servlet, the servlet is able to handle client requests to implement FDTD simulations. These requests are processed in the service method. It should be noted that each such request is run in its own lightweight servlet thread with the result that concurrent requests are catered for. The three main processing stages, namely evaluation of specification parameters, FDTD engine calling and data post-processing, are implemented. At the start of these processing stages, the servlet receives two objects: the HttpServletRequest object, which is concerned with client to servlet communication, and the HttpServletResponse object which returns data result back to client. Evaluation of specification parameters received from clients is undertaken to run FDTD simulation engine through socket or extract data result according to the requirement of clients. Through socket, FDTD engine calling undertakes the communication between web server and supercomputer

which the FDTD simulation engine is in, including running the FDTD engine with relative parameters from clients and receiving data result objects. As soon as data result is available, data post-processing is implemented to get specific data, format and return them to clients. Communications between applets (clients) and servlets (web-server) and between servlets and FDTD simulation engine is described subsequently.

4.4.2 Applet-Servlet Communication with POST

In the web-based FDTD simulation system, applet provides the user with a comfortable and flexible interface. Once the applet has opened a connection to the URL, the input stream from the servlet is accessed. The applet can read this input stream and process the data accordingly. The applet can send data to the servlet by sending a POST method, because of GET's size limitation [26]-[27]. To POST data to a servlet, the java.net.URLConnection class is used. The POST method is powerful because programmers can send any form of data (plain text, binary, etc) only after setting the content type in the HTTP request header. However, the servlet must be able to handle the type of data that the applet sends.

The code fragment below shows how to send a POST method to a servlet URL. The details of transmitting the data are discussed later.

```
// connect to the servlet
URL fdtdServlet = new URL (servletLocation);
URLConnection servletConnection = fdtdServlet.openConnection();
```

```
// inform the connection that we will send output and accept input
servletConnection.setDoInput(true);
servletConnection.setDoOutput(true);
// Don't use a cached version of URL connection.
servletConnection.setUseCaches (false);
servletConnection.setDefaultUseCaches (false);
// Specify the content type that we will send binary data
servletConnection.setRequestProperty
("Content-Type", "<insert favorite mime type>");
// get input and output streams on servlet
. . .
// send your data to the servlet
. . .
```

4.4.2.1 Communicating Write/Read Object Serialization

In this simulation, it is necessary to provide a higher level of abstraction. Instead of passing each parameter of computing information (i.e. dimension, size, sources and so on) as name value pairs, it is sent as a true Java object, which encapsulates all of the information about parameters. This information is gathered from the applet and a parameter object is created. When implementing a specific simulation, designers simply send the parameters object to the servlet. Upon receipt of the parameters object, the servlet would process them and send them to the FDTD server. Furthermore, to optimize the application, data compressing and decompressing are necessary in the communication between applets and servlets.

The code fragment below shows how to serialize an object to an output stream. In this part, there is a connection to a server URL and it only needs simply serialize the object obj.

```
ObjectOutputStream out = new ObjectOutputStream (new
        GZIPOutputStream (con.getOutputStream ()));
out.writeObject (obj)  //obj is the objects will be sent to the servlet.
```

In the above code fragment, an ObjectOutputStream is created and responsible for serializing an object. The object is actually serialized when the writeObject () method is called with the target object as its parameter. On the other hand, GZIPOutputStream class is used to insert zip stream functionality into object streams for compressing data. At this time, the object including parameters information is written to the output stream.

The code fragment below shows how to deserialize an object from an input stream.

```
ObjectInputStream in = new ObjectInputStream(new
            GZIPInputStream (con.getInputStream ()));
Object obj = in.readObject ();
```

An ObjectInputStream is created based on the URL connection. Reversely with GZIPOutputStream class, GZIPInputStream class is used to decompress data received through URL connection. The object is deserialized by simply calling the readObject () method. At this point, the object is available for normal use.

As shown in the above two code fragments, Object serialization is very straightforward and convenient and can be used to pass objects back and forth between applets and servlets in the web-based FDTD system.

4.4.2.2 Sending Objects from an Applet to a Servlet

In the web-based FDTD simulation system, parameters inputted by clients are encapsulated into a vector object. Through sending a POST method to the servlet, the client-side code fragment opens a URL connection to the servlet URL, and then it sends output data—the vector object over the connection and receives input. In the web-based simulation system, the applet sends parameters object to the servlet when a new simulation is implemented. Fig 4.3 displays the object interaction between the servlet and the applet.

When the web-based simulation system is in the process of implementing a new simulation, the servlet must read object and run the new FDTD accordingly. Thus, it needs code on the server side to receive a serialized object, e.g.,

```
private void sendObjecttoServlet (URLConnection con,Object obj) throws
IOException
{
ObjectOutputStream out = new ObjectOutputStream (new
   GZIPOutputStream (con.getOutputStream ()));
   if (obj!=null)
   {
     out.writeObject (obj);
   }
   out.close ();
 }
```

4.4.2.3 Sending Objects from a Servlet to an Applet

In the simulation system, the servlet is now capable of receiving a parameter object and calling a FDTD engine to implement a new simulation, but the problem exists to do with the total number of objects. The solution is to write all the objects to a byte array and send the byte array. On the receiving end, it is necessary to get the byte array and to convert that to objects. Because the problem happens only in the direction of the servlet to the applet, it is only implemented there.

The code fragment below shows how to convert the byte array to the objects.

```
//convert the obj data to byte
  private static byte [] convertObjectToByteArray (Object obj) throws Exception
  {
    ByteArrayOutputStream b=new ByteArrayOutputStream ();
    ObjectOutputStream out=new ObjectOutputStream (b);
    out.writeObject (obj);
    return b.toByteArray ();
  }
```

Now, the servlet must return a result data. The result data is returned as a vector of object. This interaction is also illustrated in Fig 4.3. When the servlet returns the vector of result data, there is no need to iterate through the vector and serialize each object individually. The servlet can simply serialize the entire vector in one step, since the class java.util.Vector also implements the java.io.Serializable interface.

```
//send data result to the applet in the user
  private void sendObjectToClient (HttpServletResponse resp,Object obj)
      throws Exception
```

```
{
  byte [] b=convertObjectToByteArray (obj);
  ObjectOutputStream out = new ObjectOutputStream (new
          GZIPOutputStream (resp.getOutputStream ()));
  out.writeObject (b);
  out.close ();
}
```

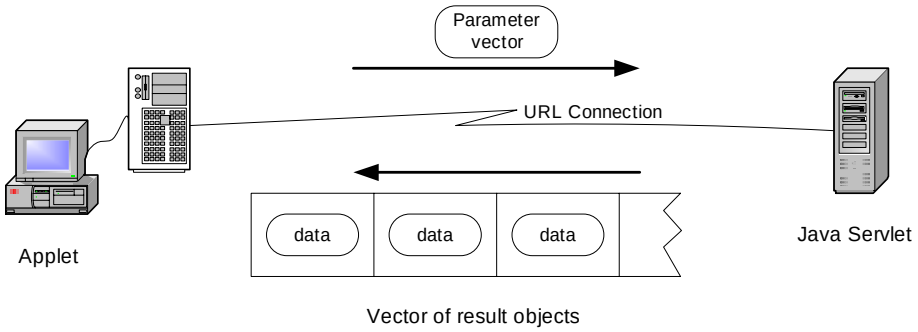

Figure 4.3: Applet-servlet object transactions

4.4.3 Communication between Web Server and FDTD Package

Multithreading socket is implemented to achieve the communication between the web server and the FDTD package. Object serialization is not limited to binary disk files. Objects can also be serialized to any output stream. This even includes an output stream based on a socket connection. So, we can serialize an object over a socket output stream and a Java object can also be deserialized or loaded from a socket input stream.

4.4.3.1 The Multi-thread Server Socket

When multiple clients access the FDTD simulation package simultaneously, multiple client requests can come into the same port and, consequently, into the same

ServerSocket. Client connection requests are queued at the port, so the server must accept the connections sequentially. However, the server can provide service to them simultaneously through the use of threads—one thread per each client connection.

The multithreaded server socket program creates a new thread for every client request. This way each client has its own connection to the server for passing data back and forth. In the web-based FDTD system, the listenSocket method loops on the server.accept call waiting for client connections and creates an instance of the ClientWorker class for each client connection it accepts.

The listenSocket method follows the basic script and the code segment is given as follows:

1. Create the server socket and begin listening.
2. Call the accept() method to get new connections.
3. Create input and output streams for the returned socket.
4. Conduct the conversation based on the agreed protocol.
5. Close the client streams and socket.
6. Go back to step 2 or continue to step 7.
7. Close the server socket.

The sample code is listed below;

```
public void listenSocket () {
   try {
     server_socket = new ServerSocket (port);
   } catch (IOException e) {
     System.out.println ("Could not listen on port"+port);
     System.exit (-1);
   }
```

```
  while (true) {
   ClientWorker w;
   try {
    w = new ClientWorker (server_socket.accept ());
    Thread t = new Thread (w);
    t.start ();
   } catch (IOException e) {
    System.out.println ("Accept failed: "+port);
    System.exit (-1);
   }
  }
 }
```

The java.lang.Runnable interface and the java.lang.Thread class are used to create and run a separate thread of execution. Any object that is to execute within a thread must implement the Runnable interface. This interface defines a single method: run(). It is the responsibility of the developer of the threaded class to provide the run() method. The operations to be performed by the thread should be done in this method. The Thread class is used to control a thread in a running program, i.e.,

```
class ClientWorker implements Runnable {
 ……
 ClientWorker (Socket client) {
  this.client = client;
 }
 public void run () {
  BufferedReader input = null;
  PrintWriter output = null;
  try {
   input = new BufferedReader (new InputStreamReader (client.getInputStream ()));
   output = new PrintWriter (client.getOutputStream (), true);
```

```
    } catch (IOException e) {
      System.out.println ("in or out failed");
      System.exit (-1);
    }
    …….
  }
}
```

4.4.3.2 A Client Socket

When a user sends a new request and simulation parameters to the web server, the servlet receives parameters and calls a new simulation through socket. The communication between the web server and FDTD simulation package through socket is encapsulated into a function: public void socketCommunicate(Vector parameter).

As soon as the servlet receives a new request and the relative parameters, it creates a new socket and tries to connect with the server socket. After the server socket accepts it, parameters are sent to the server through socket.getOutputStream(), that is,

```
    int port = 1500;
    String server =" 192.168.42.103";
    Socket socket = null;
    BufferedReader input;
    BufferedReader dataput;
    PrintWriter output;
    int ERROR = 1;
    System.out.println ("server port = 1500 (default)");
    // connect to server
    try {
      socket = new Socket (server, port);
```

```
    System.out.println ("Connected with server" + socket.getInetAddress () + ":" +
socket.getPort ());
  }
  catch (UnknownHostException e)
  {
   System.out.println (e);
   System.exit (ERROR);
  }
  catch (IOException e)
  {
   System.out.println (e);
   System.exit (ERROR);
  }
 …
  try
  {
   socket.close ();
  }
  catch (IOException e)
  {
   System.out.println (e);
  }
 }
```

After sending parameters, the client socket is waiting data results produced by FDTD package in a while() loop. Once the loop checks data are available, it reads them and is ready for processing later through socket.getInputStream(). The code segment is shown in the following.

```
dataput = new BufferedReader(new InputStreamReader(socket.getInputStream()));
…
while(true)
```

```
{
   String record=dataput.readLine ();
   if (record!=null)
   {
      if (record.indexOf ("end")>=0)
      {
         System.out.println (record);
         break;
      }
      else
      {
         fileSave.println (record); //save data in a file
         System.out.println (record);
      }
   }
}
```

4.4.4 Web-enabling FDTD

4.4.4.1 The FDTD Method

The FDTD (Finite-Difference Time-Domain) is a time domain method, and was first established by Yee as a three dimensional solution of the Maxwell's curl equations [32]. In principle, a volume of space containing any object or collection of objects is subjected to an electromagnetic disturbance, FDTD then solves for the fields throughout the volume as a function of time. It has been widely applied for various applications including RF devices, IC clips and system EMC/EMI design and analysis.

These applications can all be supported by the basic FDTD algorithm, requiring only different pre- and post-processing interfaces to obtain the final results.

The FDTD method provides a direct integration of Maxwell's time-dependent equations while the paralleling FDTD is to incorporate parallelism into the solution of Maxwell's equations. Using this method can simulate the signal propagation of the differential lines that possess the high signal transmission speed.

The formulations of FDTD method begin by considering differential forms of Maxwell's two curl equations which govern the signal propagation of fields in the structures. For the sake of simplicity, the media are assumed to be piecewise uniform, isotropic, and homogeneous and the structure is assumed to be lossless. With these assumptions, Maxwell's curl equations can be written as

$$\mu \frac{\partial H}{\partial t} = -\nabla \times E \qquad (1)$$

$$\varepsilon \frac{\partial E}{\partial t} = \nabla \times H \qquad (2)$$

In order to find an approximate solution to this set of equations, the problem is discretized over a finite three-dimensional computational domain with appropriate conditions enforced on the source, conductors, and mesh walls.

To obtain discrete approximations to these continuous partial differential equations, the central difference approximation is used on both the temporal and spatial first-order partial differentiations in well-known Yee's cells as shown in Fig 4.4 [32]. The entire computational domain is obtained by stacking these rectangular cubes into a larger rectangular volume, where $\hat{x}$, $\hat{y}$, $\hat{z}$ dimensions are Δx , Δy , Δz , respectively.

The advantages of this field arrangement are that the central differences are realized in the calculation of each field component and the continuity of tangential field components is automatically satisfied. Because there are only six unique field components within the unit cell, the six field components in the lowest locations of the unit cell in Fig 4.4 are considered to be a unit node with subscript indices *i*, *j*, and *k* corresponding to the node numbers in the $\hat{x}$, $\hat{y}$ and $\hat{z}$ directions.

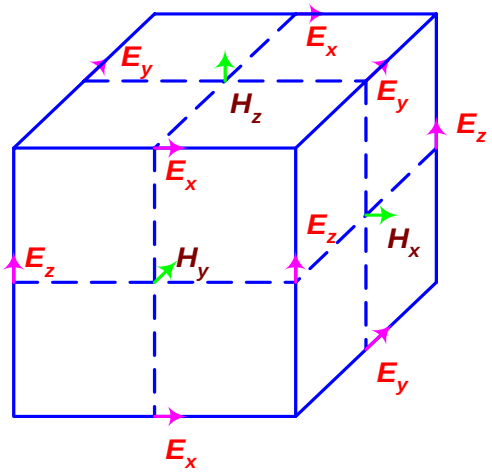

Figure 4.4: Yee's cell in 3D space

This notation implicitly assumes the $\pm 1/2$ space indices and thus simplifies the notation, rendering the formulas directly implementable on the computer. The time steps are indicated with the superscript n. Using this field component arrangement, the above notation, and the central difference approximation, the explicit finite difference approximations to equations (1) and (2) are

$$E_x^{n+1}(i,j,k) = E_x^n(i,j,k) + \Delta t/\varepsilon \left[H_z^{n+1/2}(i,j+1,k) - H_z^{n+1/2}(i,j,k)/\Delta y - H_y^{n+1/2}(i,j,k+1) - H_y^{n+1/2}(i,j,k)/\Delta z\right] \quad (3)$$

$$E_y^{n+1}(i,j,k) = E_y^n(i,j,k) + \Delta t/\varepsilon \left[H_x^{n+1/2}(i,j,k+1) - H_x^{n+1/2}(i,j,k)/\Delta z - H_z^{n+1/2}(i+1,j,k) - H_z^{n+1/2}(i,j,k)/\Delta x\right] \quad (4)$$

$$E_z^{n+1}(i,j,k) = E_z^n(i,j,k) + \Delta t/\varepsilon \left[H_y^{n+1/2}(i+1,j,k) - H_y^{n+1/2}(i,j,k)/\Delta x - H_x^{n+1/2}(i,j+1,k) - H_x^{n+1/2}(i,j,k)/\Delta y\right] \quad (5)$$

$$H_x^{n+1/2}(i,j,k) = H_x^{n-1/2}(i,j,k) + \Delta t/\varepsilon \left[E_y^n(i,j,k) - E_y^n(i,j,k-1)/\Delta z - E_z^n(i,j,k) - E_x^n(i,j-1,k)/\Delta y\right] \quad (6)$$

$$H_y^{n+1/2}(i,j,k) = H_y^{n-1/2}(i,j,k) + \Delta t/\varepsilon \left[E_z^n(i,j,k) - E_z^n(i-1,j,k)/\Delta x - E_x^n(i,j,k) - E_x^n(i,j,k-1)/\Delta z\right] \quad (7)$$

$$H_z^{n+1/2}(i,j,k) = H_z^{n-1/2}(i,j,k) + \Delta t/\varepsilon \left[E_x^n(i,j,k) - E_x^n(i,j-1,k)/\Delta y - E_y^n(i,j,k) - E_y^n(i-1,j,k)/\Delta x\right] \quad (8)$$

The half time steps indicate that *E* and *H* are alternately calculated in order to achieve central differences for the time derivatives. In these equations, the permittivity and the permeability are set to the appropriate values depending on the location of each field component. For the electric field components on the dielectric-air interface, the average of two permittivities, $(\varepsilon_0 + \varepsilon_1)/2$, is used.

Due to the use of central differences in these approximations, the error is second order in both the spatial and temporal steps: i.e., if Δx, Δy, Δz and Δt are proportional to Δl, then the global error is $O(\Delta l^2)$. The time step that may be used is limited by the stability restriction of the finite difference equations,

$$\Delta t \le \frac{1}{\tau_{max}} \frac{1}{\sqrt{1/\Delta x^2 + 1/\Delta y^2 + 1/\Delta z^2}} \qquad (9)$$

where $\tau_{\max}$ is the maximum velocity of light in the computational volume. Typically, $\tau_{\max}$ will be the velocity of light in free space. The entire volume is filled with solution of *E(r, t)* and *H(r, t)* in the volume of the computational domain or mesh; however, special consideration is required for the source, the conductors, and the mesh walls.

4.4.4.2 Simulation Example

This section takes a dipole antenna as a simulation example to examine the web-based FDTD system. The steps will be described subsequently.

(1) Drawing the geometries and setting material parameters

In this step, the geometry of the problem for full-wave analysis will be created and then the material parameters are set for the objects drawn. In this problem, only a dipole needs be input with perfect electric conductive material, as shown in Fig 4.5.

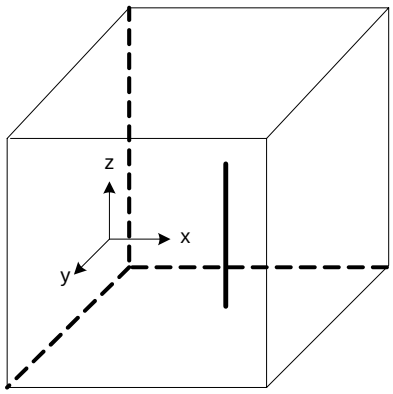

Figure 4.5: The geometric model of the dipole problem

(2) Meshing the geometries

Once the geometries are built up, they will be meshed into a large number of elements with meshing solver. In the FDTD method, the computational region is stacked by millions of the FDTD cells shown in Fig4.4. In each cell, electric and magnetic field components are defined, and are calculated in the solver with a system of iteration equations. The cell parameters pertaining to the property of material occupied by the cell are calculated and stored in a text file for later use by the solver.

(3) Setting sources and the outputs needed

In this step, the number and the type of sources, its polarization and placement, time-domain waveform and corresponding parameters need to be specified. The types of source include the current source, voltage source, and plane incident wave source with various time-domain waveforms, such as Gaussian pulse. Also, the result data, which is required to be output after the simulation, must be specified. These output could include time-domain and frequency-domain currents and voltages, the impedances, far fields as well as near field in frequency domain and time domain. A text file named project file is produced and list those messages. This file will be read by the solver. The other parameters are also specified in this file, including the size of computational space, the number of iteration in time domain, the size of FDTD cell, the directory and name of mesh file.

For this example, a computational region of $41 \times 41 \times 61$ cells is used with the size of each cell $\Delta x = \Delta y = \Delta z = 1cm$. A voltage with time-domain Gaussian pulse is added in the middle of the dipole with the z polarization. In this dipole simulation, the

outputs include radiation pattern of this EM problem, time-domain and frequency-domain current and voltage, and impedance in the middle of the dipole.

(4) Generating a solution

The FDTD solver first reads the project file. From this file, it gets the name and the directory of mesh file. Then, the solver reads mesh file and initializes computational region with the data in this file. The solver also gets basic parameters, the messages about sources and required output in the calculation. By means of those messages, the FDTD iterations are built up, and 3D time-domain electromagnetic fields inside the computational region are obtained. The solver also produces the required data for output in terms of those 3D time-domain fields, which will be recorded in a text file for post-processing.

In this problem, radiation pattern, time-domain and frequency-domain current and voltage, and impedance in the middle of the dipole are stored in the output file.

(5) Analyzing and visualizing the solution

In this step, the output results from the solver are analyzed, and visualized with graphical plots. For this problem, the time-domain current and voltage are presented in Fig 4.6 and Fig 4.7, respectively.

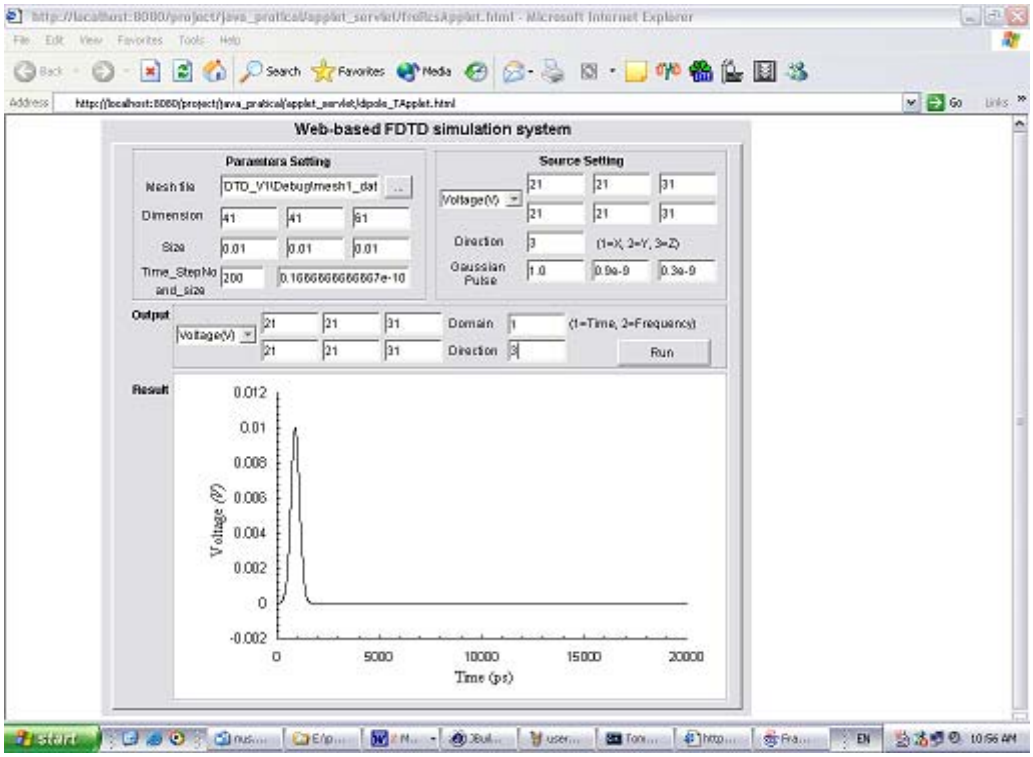

Figure 4.6: Transient voltage waveform simulated via web-based FDTD system

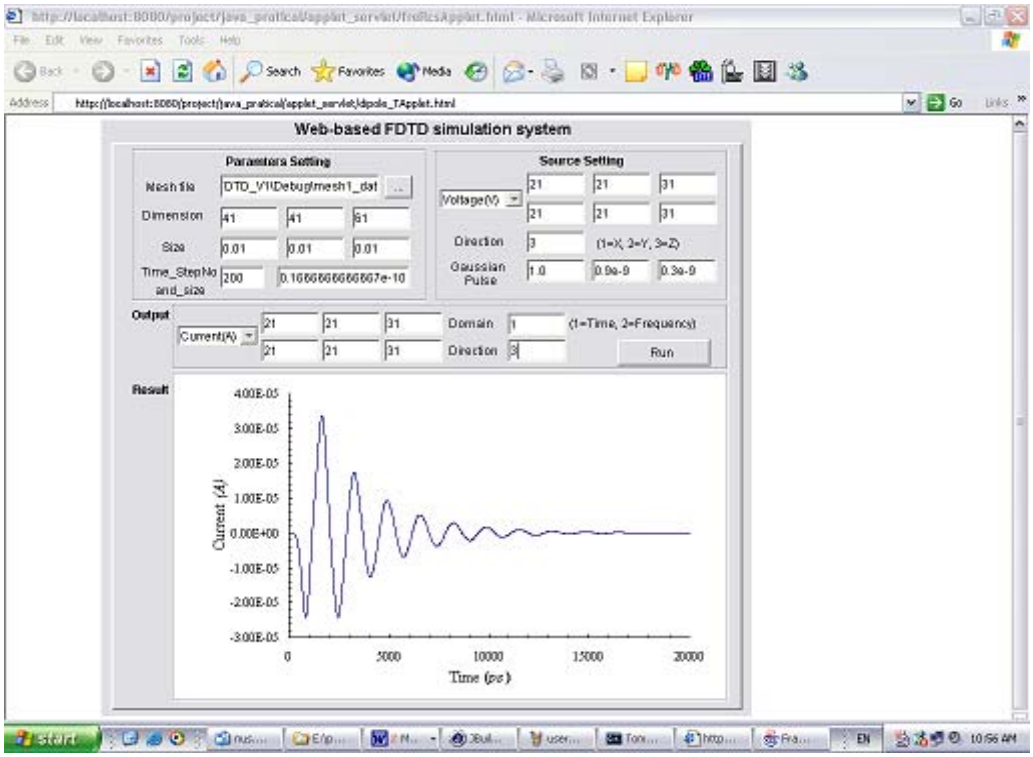

Figure 4.7: Transient current waveform simulated via web-based FDTD system

4.4.5 Conclusions

Even though there are many ways to achieve communication between applet and servlet, but because of the feature of HTTP object streams, it is used to overcome the applet security problem and to achieve the communication between applet and servlet. On the other hand, Multithreading socket is implemented to achieve the communication between web server and FDTD package, and makes FDTD accessible remotely.

With the above techniques, the Web-based FDTD simulation system is developed to provide the sophisticated FDTD simulation in an online environment. Users can run FDTD simulations anytime, from almost any computer connected to the Internet.

Chapter 5 The Study of Interface between Java and VRML

In this chapter, the interface and integration between Java and VRML, and its application in the development of web-based EM simulation is presented and discussed. External Application Interface (EAI) is described to achieve the integration of Java and VRML.

5.1 Introduction

Visualization is an important part of the EM simulation software modules. It can be achieved through commercial packages such as Matlab and AVS (Advanced Visual Systems), or the Java graphic libraries. VRML is also now available on the WWW browsers and modules can be organized as virtual environments, providing a new powerful way to view complex data sets where the user is completely embedded in the scene.

To visualize EM simulation results remotely through the Internet, the concept of the interaction of Java and VRML worlds can be extended in the domain of EM simulation. Some EM simulation packages can directly generate results into VRML files, which have an extension .wrl and permit a better interactive 3D visualization of represented models. The 3D visualization of the simulated EM problem can be achieved through VRML description, which should be generated, and controlled from applets.

To increase EM simulation results' accessibility, the methodology for visualizing the results of EM simulation on the Internet relies on the collaborative power of Java and

VRML, which makes it available world-wide through any VRML97 compatible web browser, such as CosmoPlayer. It allows a user, located anywhere, to access and visualize the simulation results through its URL. To use the VRML EAI (External Application Interface) to control the process design flow through a dedicated Java applet, the client will be built as a java applet running on the web browser connected to the VRML scene through EAI [28]-[31].

5.2 VRML

Paralleling the advances in web technology, advances in computing technology provide capabilities that were until recently beyond the reach of machines generally available. As evidence and in support of these advances, a 3D interactive visualization language, the Virtual Reality Modeling Language (VRML) has been standardized and is widely available. VRML offers opportunities for demonstrating the value of distributing 3D web content over the World Wide Web (WWW).

The visualization is an important part of the web-based EM simulation modules. To view VRML models of EM simulation is now available on the web browsers and the modules can be organized as virtual environments, providing a new powerful way to visualize complex EM simulation data sets where the user is completely embedded in the scene.

5.2.1 Overview

VRML is a file format for describing interactive 3D objects to be experienced on the web and the current standard for interactive 3D descriptions of objects. It defines a set

of objects useful for 3D graphics, multi-media, and interactive objects building. These objects are called nodes, and contain elemental data, which is stored in fields and events. The main goal of the current version of VRML 2.0 is to provide a rich 3D interactive graphical environment, allowing the user to define static and animated worlds, and to interact with them. The capabilities of navigation and viewpoints are built into VRML and, thus, can be used as a graphical display engine. VRML inherently supports an event driven model, which allows routing of the field values inside the nodes to other values thus changing the scene. All the interactive elements of the VRML model are named, and can be directly manipulated by the Java applet.

VRML has been designed to fulfill the following requirements:

- Authorability

  Enable the development of computer programs capable of creating, editing, and maintaining VRML files, as well as automatic translation programs for converting other commonly used 3D file formats into VRML files.

- Composability

  Provide the ability to use and combine dynamic 3D objects within a VRML world and thus allow re-usability.

- Extensibility

  Provide the ability to add new object types not explicitly defined in VRML.

- Be capable of implementation

Capable of implementation on a wide range of systems.

- Performance

Emphasize scalable, interactive performance on a wide variety of computing platforms.

- Scalability

Enable arbitrarily large dynamic 3D worlds.

5.2.2 Interface Components

5.2.2.1 Concepts

There are three main items in a VRML browser that can be accessed from an external application: The browser, nodes within the scene graph and fields within nodes [29]. The definition and specifications are framed in terms of services. A VRML browser exposes a set of services which allow external applications to interact with it.

5.2.2.2 Application

An application is the external process that is not implicitly part of the VRML browser. This application makes some form of connection to the VRML browser along which requests are made of the browser. The application does not exist as part of the VRML

browser nor forms part of the VRML execution model. An application may reside on another machine from the VRML browser.

5.2.2.3 Session

A session defines the life of a single connection between the application and the VRML browser. It is possible for a single browser to be servicing multiple sessions simultaneously. It is also possible that a single application may contain a number of separate sessions to multiple browsers. Multiple simultaneous sessions between external applications and multiple VRML browsers are permissible. However, individual implementations may place some restrictions on such multiple simultaneous sessions.

A session is not an implementable part of this specification. It is purely a conceptual mechanism by which the services can make requests for services. It may exist prior to any connection being established between a browser and external application (e.g. CORBA ORB) or is established simultaneously with the request for a browser connection (instantiating a Java Browser object instance).

5.2.2.4 Browser

The browser is the basic encapsulation mechanism for an active VRML scene graph (that is one where time is progressing—not as a file stored on disk). As it contains the entire scene graph, it also provides a core set of capabilities for dynamically manipulating that scene graph at a coarse level. For example, it allows users to dynamically generate new content from a string of characters.

A user may have many VRML browsers running simultaneously on their machine. Therefore, each browser shall be represented by a unique identifier within that session. This identifier is required to be identical for multiple requests of a single browser instance. This is to enable two applications that have access to the one browser instance to share information in an unambiguous way. Any action that requires use of the browser functionality shall identify the service request with a browser identifier.

5.2.2.5 Nodes

The smallest unit of interaction with the elements in the scene graph is called the node. The node is equivalent to the VRML node. A node can be removed as a unit from the scene graph, stored and then re-inserted at another position at some later time in the same session without detrimental effect.

Each node is defined by an unique identifier. This identifier is unique for that session. That is, it is possible that a single browser may be servicing multiple applications simultaneously and therefore all node identifiers are unique and invariant for the life of the session. This allows two external applications to potentially share data between them unambiguously and still has either make service requests of the browser with that shared data.

It is not possible to directly manipulate a node's properties as separate entities to the node itself. Most operations in the EAI begin by obtaining a reference to a node. To reference a node it must be named using the DEF construct. Once a reference is obtained, the eventIns and eventOuts of that node can be accessed. Since an

exposedField implicitly contains an eventIn and an eventOut, these are accessible as well using the field name or with the set_ and _changed modifiers.

Node identifiers may also be used to represent an empty node. Fields representing nodes (SFNode and MFNode) require the ability to clear the field setting it to NULL. Representing an empty SFNode field value is with a NULL value. Representing an empty MFNode field is implementation dependent.

There are many kinds of nodes in VRML, including geometric nodes, illumination nodes, grouping nodes, particularly important for animated worlds, sensor nodes, interpolator nodes and script nodes [30]. They will be described subsequently.

● Sensor nodes

Sensor nodes generate events based on user actions; the TouchSensor node, for example, detects when the user clicks the mouse over a specified object (or group of objects), and generates an eventOut. This eventOut may be routed to other eventIn(s), and cause the start of an animation. Sensors are responsible for the user interaction in VRML. Furthermore, they are not restricted to generate events based on user actions; the TimeSensor, for instance, automatically generates an event at each tick of the clock, being normally used as the animation clock.

● Interpolator nodes

Interpolator nodes define the keyframes of an animation, interpolated by a linear function. An example is the PositionInterpolator node, where the user defines n key positions and n time instants (each instant associated to a key position). Used in

conjunction with a TimeSensor, the PositionInterpolator generates an event at each clock tick, representing the current position, resulted from the interpolation function.

The events generated by sensor and interpolator nodes routed to geometric, illumination, and grouping nodes may define interesting keyframe animations. Nevertheless, the routing approach is limited, since it does not allow the handling of a whole class of behaviors that depends on logic operations. VRML overcomes this limitation defining a special node, called Script, which allows the user to connect the animation to a program such as Java program, where the events can be processed.

● Script nodes

Interfaces between VRML and Java are effected through Script nodes, an event engine, DEF/USE naming conventions, and ROUTEs connecting various nodes and fields in the VRML scene. VRML provides the 3D scene graph, Script nodes encapsulate Java functionality, and ROUTEs provide the wiring that connects computation to rendering.

Script nodes appear in the VRML file, encapsulating the Java code and providing naming conventions for interconnecting Java variables with field values in the scene. (Similar scripting conventions are specified for JavaScript). Interfaced Java classes import the vrml.* class libraries in order to provide type conversion (for both nodes and simple data types) between Java and VRML. Java classes used by Script nodes must extend the vrml.node.Script class in order to interface properly with the VRML browser. The basic interface and a good description of Script nodes are excerpted from the (VRML 97) specification bellow:

```
Script {
exposedField MFString url []
field          SFBool    directOutput FALSE
field          SFBool    mustEvaluate FALSE
# And any number of:
eventIn        eventType    eventName
field          fieldType    fieldName    initialValue
eventOut       eventType    eventName
}
```

Script node is used to program behavior in a scene. Script nodes typically signify a change or user action, receive events from other nodes, contain a program module that performs some computation, and effect change somewhere else in the scene by sending events.

5.2.2.6 Fields

Within nodes are individual fields. While it is not possible to directly manipulate a node, a field is the method of direct manipulation of individual properties.

The access granted to individual fields is defined by the VRML specification. For example, it is not possible to change the value of a "field" field. "exposedField"s may have values written to them and values read from them. A field is assigned a field identifier. This is non-unique and requires a node ID plus field ID to specify a particular field with which to interact.

Fields may be read or written at any time during the course of the session. An application may register and unregistrer to receive notification of when values of the field changes. During the registration process the application can supply a token that

will be returned along with the data value of the event. This token can be used by the application to uniquely identify this event in cases where events are not implicitly unique. The token is not required to be passed along with the service request and may be kept as part of the internals of the implementation on the application interface.

Any eventOut of a node to which the application has a reference can be read. The value read is the last value sent to that eventOut or the default value for that eventOut type if no event has ever been sent. The data read is specific to the field type of that eventOut and is formatted appropriate to the language or protocol used.

VRML scenes are based on nodes defining a scene graph. Each node defines a name, a type, and default values for its parameters. There are two kinds of parameters: fields and events. Fields can be called simply “fields” (private) or “exposedFields” (public). Events can be sent from a node to another by an “eventOut” parameter and received by an “eventIn”. Events signalize changes caused by external stimuli and can be propagated by the node using Routes, which connect an eventOut to an eventIn of the same type (see Fig 5.1). Events and Routes drive the animation of the worlds.

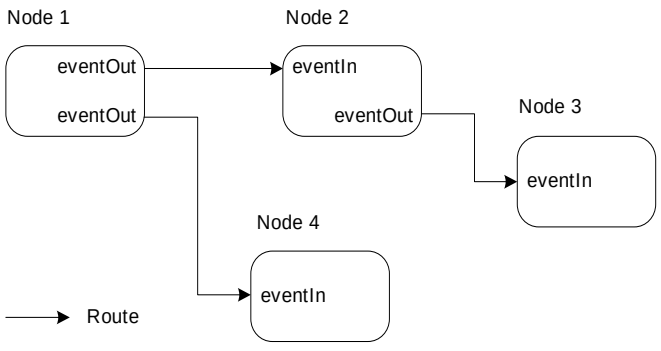

Figure 5.1: Nodes connected by routed events

To communicate with the real world, a programming language is needed to line VRML with the real world. This program can be written in any programming language, but Java is considered here, since it is presently by far the most used on based on the web. Therefore, this requires that the VRML browser supports the Java-VRML interface. The External Authoring Interface (EAI) is often used to link Java and VRML.

5.3 Integration of VRML and Java

With VRML files generated by EM simulator, the integration of VRML and Java is for the interactive control of visualization of simulation results. The combination of VRML and java is a very powerful tool for the control an animations behavior [31]. Interactive animations that have their behavior defined by user actions can be modeled with VRML using a Script node and its associated program, which is capable of receiving parameters from another java application, responsible for the user interface.

All the interactive elements of the VRML model are named, and can be directly manipulated by the Java applet. The combination of VRML and java for the construction of highly interactive animations can be used, whose behavior is defined in real-time by user's actions. Java is responsible to the interpretation of user's generated events form the mouse and keyboard and captured by the browser. The animations are modeled in VRML, which allows the definition of a java program to process and generate events that determine the behavior of scene elements.

Java servlets can interact with and manage a VRML model, and enable the integration of java programs that run on the web server. The servlets can manage information from the client browser that reside on the server and can, in addition, produce web content of any type. Servlets provide three important capabilities. First, servlet calls make possible the logging of information generated by a VRML application. Second, servlets can restore the work from a prior session by retrieving information from the log file and emitting the appropriate VRML content. Third, servlets can be used to generate models specified parametrically.

5.3.1 Java 3D and VRML

Recently, Sun announced its intention to create a 3D toolkit for Java, called Java3D, some time in the future. This has caused a good deal of confusion about how 3D content should be added to a Java application. Java3D will use traditional programming techniques to create and control a 3D presentation. Using Java3D, a programmer would create and compose 3D objects, or read static objects from a file. Code to control the behavior of these objects would then be written. This is a very powerful model of 3D content creation, but it is also extremely tedious. It requires the programmer to understand the structure of any object imported, in order to apply behavioral control in the proper places. This would require a programmer with sophisticated graphic design abilities, a graphic designer with highly technical programming skills, or extremely tight communication between a programmer and a graphic designer.

In contrast, VRML provides a high level mechanism for communication between the objects and the Java controlling them. The graphic designer can create an object with

built-in animation, and then provide a high level mechanism for controlling the object. The programmer can then control the object by simply communicating with the provided interface. This allows many graphic designers to work on different parts of the project. One or more programmers would provide the complex behavior and would create a controlling Java applet. Finally, a technical director would put all the pieces together. This allows a team with very specific skills to work together in producing a large web application.

A toolkit approach to 3D content is useful for some applications, such as scientific visualization of some new phenomena, where a fully custom approach is appropriate. But, for the large majority of today's web projects, the marriage of Java and VRML provide the perfect solution.

5.3.2 Integration with JSAI

Java can be used with VRML in two ways. VRML has a Script node and Java can be used as the scripting language for that node. The Java script interacts with the VRML world through the Java Script Authoring Interface (JSAI). The JSAI allows Java to send events to other nodes in the VRML world, create new scene components, and query for information about the scene [c]. The Java script receives events from other nodes, which stimulates it into execution, where it can perform algorithms, or utilize other Java packages (such as the Network package). The script can then send the results of this execution to other nodes as events. This allows Java to provide complex behaviors to the objects in a scene. VRML has a prototyping mechanism which allows the nodes of a scene and the controlling Java script to be packaged together and composed into other scenes.

While the JSAI is included as an optional component of the VRML specification, CosmoPlayer does not yet support this feature. Future versions will have full support for both interfaces, but for now the EAI is the only Java interface supported by CosmoPlayer.

5.3.3 Integration with EAI

Java can control a VRML world externally, using the External Authoring Interface (EAI). The EAI allows one or more VRML worlds to be added to a Java applet or application. This allows VRML to be part of a larger multimedia web presentation. The EAI allows much of the same functionality as the JSAI. You can send events to VRML nodes, create new nodes, and query the VRML world about its state and characteristics. But rather than being stimulated by an event coming into a Script node, the EAI allows a currently running Java applet to control a VRML world, just like it would control any other media. For example, pressing an AWT widget under control of the Java applet could cause an event to be sent to the VRML world which would change the color of a sphere, the height of a bar graph, or the price readout of a stock.

5.4 External Application Interface (EAI)

EAI (External Application Interface) is an interface that allows an external program to access the nodes in a VRML scene using the existing VRML event models. Here the external program is developed in Java.

The Java-VRML interaction takes place through the EAI, which gives the functionality to extend the features of VRML by adding the power of the

programming language Java. The communication takes place with the browser plug-in interface that allows embedded objects on a web page to communicate with each other.

EAI enables a Java program to modify VRML scenery directly inside the program and thus extends the features of VRML. It allows programs written in Java and JavaScript to control the contents of a VRML world. The controls available include object creation and removing, operations as rotation, translation, user's point of view changing, and properties changing, etc. The VRML EAI can be used to control the visualization of EM simulation results through a dedicated Java applet. The client is built as a Java applet run on the web browser connected to the VRML scene through EAI. Fig 5.2 depicts the relationship between the Java Applet, the EAI and the VRML Scene [31]. The EAI provides the API to allow the Java applet to interact with the VRML scene.

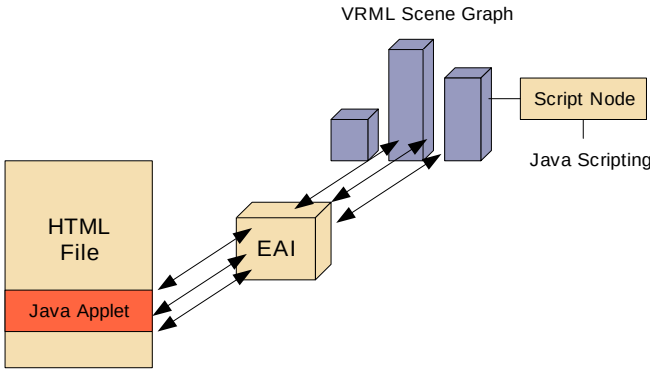

Figure 5.2: The diagram of Java Applet-VRML with EAI

Conceptually, the EAI allows four types of access into the VRML scene:

● Accessing the functionality of the Browser Script Interface,

● Sending events to eventIns of nodes inside the scene,

● Reading the last value sent from eventOuts of nodes inside the scene, and

● Getting notified when events change values of node fields inside the scene.

The EAI is also a Java API that is utilized in the same way as any other Java class API. These classes are shipped with an EAI enabled VRML browser, rather than being part of the standard JDK runtime environment. EAI classes are included in the vrml.external.* packages. These packages contain all the classes and methods needed to access the VRML world, send events to it, and register methods to be notified when an event is generated by the world.

5.5 Conclusion

The integration of VRML and Java and its application in the development of web-based EM simulation system is discussed in this chapter. Through EAI, the integration of Java and VRML can be achieved to interactively visualize simulation results on line. Through EAI, the Java-VRML interaction gives the functionality to extend the features of VRML by adding the power of the programming language Java. Integrating VRML and Java makes the web-based cooperative electromagnetic simulation possible.

Chapter 6 Conclusion and Future Work

In this chapter, conclusions on the development of web-based electromagnetic simulation system, including main techniques used in its development, are summarized. Following that, future development of EM simulation system on line, including integration of Java and VRML as well as future work are discussed.

6.1 Conclusions

This project takes advantage of web and associated techniques to develop the web-based electromagnetic simulation system. As most electromagnetic simulation packages only run at X Windows, the work carried out in this project overcomes requirements of high capacities and high performance computers, and therefore lower the cost of maintaining these computers. It enables designers to simulate complex electromagnetic problems and to visualize the results at different platforms at different locations. Web application techniques, including socket communication, multithreading programming, applet, and servlets technique have been applied in this project.

This project makes full use of features and components of Java techniques to make electromagnetic simulation packages accessible and portable across very different platforms on the World Wide Web. A simple and convenient way, via HTTP object streams, achieves the communication between applet and servlet, and overcomes applet security problem.

The socket technique is used to implement the communication between the web server and high performance computer in which electromagnetic simulation packages are running. Multithreading technique has been used to implement multi-access to the electromagnetic simulation packages, making multiple access to the electromagnetic simulation packages possible.

Some examples have been provided to describe the development of the web-based electromagnetic simulation system. For example, the Web-based HFSS system can provide the sophisticated HFSS in an online environment. Simulation and analysis essential to the engineering community can be provided on the system. Designers can run these applications anytime, from almost any computer connected to the Internet. Instead of tying up their own computer for days or even weeks of computations, their jobs run on our servers. This free up their computers for other tasks.

There are enormous advantages for the web-based EM simulation system. The software becomes more accessible because there is no need for the huge capital investment on high-performance and large storage computing resources. Also, less time is needed on setting up, upgrading and maintaining the software as opposed to traditional methods. Finally, the need for training is minimized in many cases.

The work carried out in this thesis has allowed the users easy access to simulation packages at any time, any place on any device. Internet is a powerful tool for engineering community. It makes information sharing extremely easier and more readily available.

6.2 Future Development and Work

The study about integration of VRML and Java and its application in the development of web-based EM simulation system, has been completed and presented in the thesis. The interface between Java and VRML has been studied to interactively visualize simulation results of a vrml file remotely. The Java-VRML interaction takes place through the EAI, which gives the functionality to extend the features of VRML by adding the power of the programming language Java. Integrating VRML and Java through EAI can achieve the web-based cooperative electromagnetic simulation.

A great deal of implementation work about the integration is now in progress. VRML and Java are powerful software languages for 3D modeling, general computation and network access. They are well matched, well specified, publically available and portable to most platforms on the Internet. VRML scenes in combination with Java can serve as the building blocks of cyberspace. Building large-scale internet worked worlds now appears possible. Using VRML and Java, practical experience and continued success will move the field of web-based cooperative EM simulation of research onto desktops anywhere, creating the new concept of EM simulation.

Now, what the EM simulation systems need is online real-time design, analytical information sharing, and collaborative EM simulation. With on-line collaboration platform, it is possible to reuse independently constructed pieces of a simulation and have them combined to solve a larger and more sophisticated EM simulation problem on line. It is recommended that the future work will be focus on the development of EM simulation system based on on-line collaboration platform.

References

[1] K.L.Wu, J.Litva, R.fralich, and C.Wu, "Full wave analysis of arbitrarily shaped line-fed microstrip antennas using triangular finite-element method, "IEE proceedings-H, Vol.138, pp.421-428, Oct., 1991.

[2] Allen Taflove, *Advance in computational electromagnetics: the Finite-Difference Time-Domain Method:* Artech House. 1998.

[3] Jensen, M.A.; Rahmat Samii, Y.," Finite-difference and finite-volume time-domain techniques: comparison and hybridization", Antennas and Propagation Society International Symposium, AP-S, Vol.1, pp.108 -111, Aug, 1996.

[4] A.Christ, H.Hartnagel, "Three-dimensional finite-difference method for the analysis of microwave-device embedding," IEEE Trans. Microwave Theory Tech., Vol.35, pp.688-696, Aug, 1987.

[5] The Java™ Language Environment: A White Paper by James Gosling & Henry McGilton. http://sunsite.ee/java/whitepaper/java-whitepaper-1.html

[6] Bill Brogden, "Java™ Developer's Guide to Servlets and JSP—Build A Powerful, Dynamic Web Site With Pure Java Technology": Bpb Publications, pp. 305-335.

[7] Trail: Creating a GUI with JFC/Swing. http://java.sun.com/docs/books/tutorial/uiswing/.

[8] "*Advanced Object Serialization*" By John Zukowski, August 2001. http://developer.java.sun.com/developer/technicalArticles/ALT/serialization/.

[9] Java Cryptography Extension (JCE) for the Java 2 SDK, Standard Edition, v 1.4. http://java.sun.com/products/jce/index-14.html.

[10] "*Applet-to-Servlet Communication for Enterprise Applications*" by Duane K. Fields. http://developer.netscape.com/viewsource/fields_servlet/fields_servlet.html.

[11] Alexandros Paramythis, "Basic Networking and Multithreading", Informatics 3, 2002. http://cblinux.fhs-agenberg.ac.at/~aparamyt/ws_02/Lectures/NotesSet_7/NotesSet_7-a4.pdf

[12] "Writing multithreaded Java applications" by Alex Roetter. http://www-106.ibm.com/developerworks/library/j-thread.html.

[13] "Multi_threading" http://www.sharjah.ac.ae/courseware/electrical_eng/qassim/Network_Programming/java_thread_web2.PDF.

[14] "Multithreading in Java", The Java™ Language Environment: A White Paper by James Gosling & Henry McGilton. http://sunsite.ee/java/whitepaper/java-whitepaper-9.html.

[15] "Java Multithreading", Java Programming, BFH/HTA Biel/DUE NDIT Module I-JAV.d 1999. http://www.isbiel.ch/~due/ndit/i-jav.1/99/slides/threads.pdf.

[16] *Ansoft HFSS Documents—3D EM simulation software for RF & wireless design*, released by Ansoft Corporation, January 2001.

[17] Tarantella® Enterprise 3™ Software—A Technical Overview: A Tarantella White Paper, TTATP/30/techoverwp.doc/1.12 July 2002, http://www.tarantella.com/.

[18] R.H. Turrin, “Dual Mode Small-Aperture Antennas”, IEEE Transactions on Antennas and Propagation, March 1967.

[19] Jason Hunter and William Crawford, “Java Servlet Programming”, ISBN 1-56592-391-XE, First edition, O'Reilly Releases, published October 1998.

[20] Applet to Servlet Communication by Larry O'Brien, http://archive.devx.com/upload/free/features/javapro/1999/05may99/lo0599/lo0599.asp.

[21] Michael Morrison, Jerry Ablan and so on, “Java 1.1 Unleashed”: Java Socket Programming, Macmillan Computer Publishing USA,http://hplasim2.univ-lyon1.fr/c.ray/bks/java/htm/ch26.htm

[22] “Socket Communications”, Essentials of the Java Programming Language, Part 2.http://developer.java.sun.com/developer/onlineTraining/Programming/BasicJava2/socket.html.

[23] Java Socket Programming by Michael J. Golding, 1997.http://homepages.uel.ac.uk/2795l/pages/javaapps.htm#Top.

[24] Trail: Custom Networking http://java.sun.com/docs/books/tutorial/networking/sockets/index.html

[25] L. T. Walczowski and W.A.J. Waller, “Java servlet technology for analogue module generation”, the 6th IEEE International Conference on Electronics, Circuits and Systems, Vol.3 , pp. 1717 – 1720, 1999.

[26] Chád Darby, “Applet and Servlet Communication”, Java Developer's Journal, September 1998.

[27] Daniel J. Farkas and Narayan Murthy, Use of Applet and Servlet Communication Technique to Administer Online Examinations, Informing Science, InSITE - "Where Parallels Intersect" June 2002

[28] Kaveh E. Afshari and Shahram Payandeh, "Toward Implementation of Java/VRML Environment for Planning, Training and Tele-Operation of Robotic Systems". http://www.ensc.sfu.ca/research/erl/mobile/kaveh_paper.pdf

[29] The Virtual Reality Modeling Language (VRML)—Part 2: External authoring interface, Information technology—Computer graphics and image processing, ISO/IEC 14772-2, 1997

[30] Don Brutzman, "The Virtual Reality Modeling Language and Java", Communications of the ACM, vol. 41 no. 6, June 1998, pp. 57-64. http://www.web3D.org/WorkingGroups/vrtp/docs/vrmljava.pdf

[31] Chris Marrin, Bill McCloskey, Kent Sandvik and Don Chin, "Creating Interactive Java Applications with 3D and VRML", A Silicon Graphics, Inc. White Paper, May 1997, http://eureka.lucia.it/vrml/tutorial/eai/sgi/index.html.

[32] K. S. Yee, "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media", IEEE Trans. On Antennas and Propagat., Vol. AP-14, No. 3, pp. 302-307, May 1966.

List of Publications

1. Yun Fan, Yong-Lin Li, Er-Ping Li, Saikong Chin and Le-Wei Li, “Web-based Electromagnetic Simulation”, Proceedings of International Conference of Science Engineering Computation, pp.266-271, Dec, 2002.

2. Yun Fan, Yong-Lin Li, Er-Ping Li, Saikong Chin and Le-Wei Li, “A development of Web-based FDTD simulation system”, submitted to ICICS-PCM 2003 Secretariat Conference.

3. Yun Fan, Er-Ping Li, Yong-Lin Li and Le-Wei Li, “Development of Java Applets FDTD-Based Electromagnetic Simulation System”, Proceedings of Progress in Electromagnetics Research Symposium, PIERS 2003 in Hawaii USA, Oct. 10-13, 2003.