**Implementing Multicast Distribution through Recursive Unicast Trees**

**Abstract**

IP multicast is facing a slow take-off although it has been a hotly debated topic for more than a decade. Many reasons are responsible for this status. Hence, the Internet is likely to be organized with both unicast and multicast enabled networks. Thus, it is of utmost importance to design protocols that allow the progressive deployment of the multicast service by supporting unicast clouds. This paper presents HBH (hop-by-hop hybrid routing protocol). HBH adopts the source-specific channel abstraction to simplify address allocation and implements data distribution using recursive unicast trees, which allow the transparent support of unicast- only routers. An important original feature of HBH is its tree construction algorithm that takes into account the unicast routing asymmetries. Since most multicast routing protocols rely on the unicast infrastructure, the unicast asymmetries impact the structure of the multicast trees. We show through simulation that HBH outperforms other multicast routing protocols in terms of the delay experienced by the receivers and the bandwidth consumption of the multicast trees. Additionally, we show that HBH can be incrementally deployed and that with a small fraction of HBH-enabled routers in the network HBH outperforms application-layer multicast.

ABLE OF CONTENTS

**TITLE PAGE NO**

ABSTRACT

LIST OF TABLES

LIST OF ABBREVIATIONS

SYSTEM STUDY

FEASABILITY STUDY

EXISTING SYSTEM

PROPOSED SYSTEM

SYSTEM SPECIFICATION

SOFTWARE REQUIREMENTS

HARDWARE REQUIREMENTS

LANGUAGE SPECIFICATION

Java Technology

SYSTEM DESIGN

DFD DIAGRAMS

UML DIAGRAMS

SYSTEM TESTING AND MAINTENANCE

SYSTEM IMPLEMENTATION

SCOPE FOR FUTURE DEVELOPMENT

CONCLUSION

BIBLIOGRAPHY

APPENDIX

SCREEN SHOT

SAMPLE CODING

**SYSTEM STUDY**

**FEASIBLITY STUDY**

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

* ECONOMICAL FEASIBILITY
* TECHNICAL FEASIBILITY
* SOCIAL FEASIBILITY

**ECONOMICAL FEASIBILITY:**

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

### TECHNICAL FEASIBILITY:

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical

esources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

**SOCIAL FEASIBILITY:**

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

**EXISTING SYSTEM:**

The existing system related to Hop-By-Hop Multicast Routing Protocol is namely the EXPRESS and REUNITE protocols.

EXPRESS provides a simple solution to the multicast address allocation problem, introducing the channel abstraction that reduces the multicast conversation from to to 1 to

. A channel is identified by the pair, where the unicast is address of the source and is a class-D multicast address. The concatenation of a unicast address with a class-D address solves the address allocation problem since the unicast address is unique. The channel model introduced by EXPRESS also simplifies group management issues such as sender access control. Nevertheless, the SSM service, which was inspired by EXPRESS and standardized by the IETF, does not change the group management support provided by IGMP

REUNITE implements multicast distribution based on the unicast routing infrastructure. The basic motivation of REUNITE is that, in typical multicast trees, the majority of routers simply forward packets from one incoming interface to only one outgoing interface, because only a few routers are branching nodes. REUNITE identifies a conversation by a pair, where the unicast is address of the source and is a port number. Class-D IP addresses are not used. As receivers join the group REUNITE populates its tables to construct the distribution tree, using two control messages: and. messages travel upstream from the receivers to the source, whereas messages are periodically multicast by the source to refresh the soft-state of the tree. Only the branching nodes for the group keep entries in their MFTs. The control table, MCT, is exclusively used for tree construction, not for packet forwarding. Nonbranching routers in the tree have MCT entries for but no MFT entry.

**Limitations**

The following are the limitations of the existing system.

* does not change the group management support provided by IGMP
* The control table, MCT, is exclusively used for tree construction, not for packet forwarding
* Nonbranching routers in the tree have MCT entries for but no MFT entry
* REUNITE may fail to construct shortest-path branches in the presence of unicast routing asymmetries
* Next undesirable behavior of REUNITE is that the route for one receiver may change after the departure of another receiver. This is undesirable if some QoS mechanism is to be implemented.

**PROPOSED SYSTEM:**

The HBH multicast protocol has a tree construction algorithm that is able to better deal with the pathological cases due to asymmetric unicast routes. HBH uses two tables, an MCT and An MFT, which have nearly the same function as in REUNITE. The difference is that one entry tables in HBH stores the address of a next branching node instead of the address of a receiver, except the branching router nearest the receiver. The MFT has no entry. Data received by a branching router, , has unicast destination address set to (in REUNITE, data are addressed to). This choice makes the tree structure more stable than in REUNITE

**Advantages of Proposed system:**

* A multicast channel in HBH is identified by , where is the unicast address of the source and is a class-D IP address allocated by the source. This definition solves the address allocation problem while being compatible with SSM’s channel definition. Therefore, HBH can support IP Multicast clouds as leaves of the distribution tree.
* The tree structure of HBH has the advantage of an enhanced stability of the table entries when compared with REUNITE. The tree management scheme of HBH minimizes the impact of member departures in the tree structure.
* There is no route changes for other members when a member leaves the group because the unicast routes are symmetric.

Tree reconfiguration in REUNITE may cause route changes to the remaining receivers, as for in the example of Fig. 2. This is avoided in HBH.

**SYSTEM CONFIGURATION**

**Hardware & Software Requirements**

**Hardware specification:**

* **Monitor :** 800\*600 minimum resolution of 256 colors
* **Processor:** At least 166 MHz processor
* **Input :** Two or Three button mouse and standard 104 keyboards.

**Software specification:**

**Operating System** : Windows 98.

**Front end Tool** : Java Swing and Java Networking

**Data Bases** : SQL Server

**LANGUAGE SPECIFICATION**

**Java Technology**

Initially the language was called as “oak” but it was renamed as “Java” in 1995. The primary motivation of this language was the need for a platform-independent (i.e., architecture neutral) language that could be used to create software to be embedded in various consumer electronic devices.

* Java is a programmer’s language.
* Java is cohesive and consistent.
* Except for those constraints imposed by the Internet environment, Java gives the programmer, full control.
* Finally, Java is to Internet programming where C was to system programming.

#### Importance of Java to the Internet

Java has had a profound effect on the Internet. This is because; Java expands the Universe of objects that can move about freely in Cyberspace. In a network, two categories of objects are transmitted between the Server and the Personal computer. They are: Passive information and Dynamic active programs. The Dynamic, Self-executing programs cause serious problems in the areas of Security and probability. But, Java addresses those concerns and by doing so, has opened the door to an exciting new form of program called the Applet.

#### Java can be used to create two types of programs

Applications and Applets: An application is a program that runs on our Computer under the operating system of that computer. It is more or less like one creating using C or C++. Java’s ability to create Applets makes it important. An Applet is an application designed to be transmitted over the Internet and executed by a Java –compatible web browser. An applet is actually a tiny Java program, dynamically downloaded across the network, just like an image. But the difference is, it is an intelligent program, not just a media file. It can react to the user input and dynamically change.

#### Features of Java Security

Every time you that you download a “normal” program, you are risking a viral infection. Prior to Java, most users did not download executable programs frequently, and those who did scan them for viruses prior to execution. Most users still worried about the possibility of infecting their systems with a virus. In addition, another type of malicious program exists that must be guarded against. This type of program can gather private information, such as credit card numbers, bank account balances, and passwords. Java answers both these concerns by providing a “firewall” between a network application and your computer.

When you use a Java-compatible Web browser, you can safely download Java applets without fear of virus infection or malicious intent.

#### Portability

For programs to be dynamically downloaded to all the various types of platforms connected to the Internet, some means of generating portable executable code is needed .As you will see, the same mechanism that helps ensure security also helps create portability. Indeed, Java’s solution to these two problems is both elegant and efficient.

#### The Byte code

The key that allows the Java to solve the security and portability problems is that the output of Java compiler is Byte code. Byte code is a highly optimized set of instructions designed to be executed by the Java run-time system, which is called the Java Virtual Machine (JVM). That is, in its standard form, the JVM is an interpreter for byte code.

Translating a Java program into byte code helps makes it much easier to run a program in a wide variety of environments. The reason is, once the run-time package exists for a given system, any Java program can run on it.

Although Java was designed for interpretation, there is technically nothing about Java that prevents on-the-fly compilation of byte code into native code. Sun has just completed its Just In Time (JIT) compiler for byte code. When the JIT compiler is a part of JVM, it compiles byte code into executable code in real time, on a piece-by-piece, demand basis. It is not possible to compile an entire Java program into executable code all at once, because Java performs various run-time checks that can be done only at run time. The JIT compiles code, as it is needed, during execution.

#### Java Virtual Machine (JVM)

Beyond the language, there is the Java virtual machine. The Java virtual machine is an important element of the Java technology. The virtual machine can be embedded within a web browser or an operating system. Once a piece of Java code is loaded onto a machine, it is verified. As part of the loading process, a class loader is invoked and does byte code verification makes sure that the code that’s has been generated by the compiler will not corrupt the machine that it’s loaded on. Byte code verification takes place at the end of the compilation process to make sure that is all accurate and correct. So byte code verification is integral to the compiling and executing of Java code.

**Overall Description:**

# Java Source

## *Java byte code*

# JavaVM

**Java**

**.Class**

#### Picture showing the development process of JAVA Program

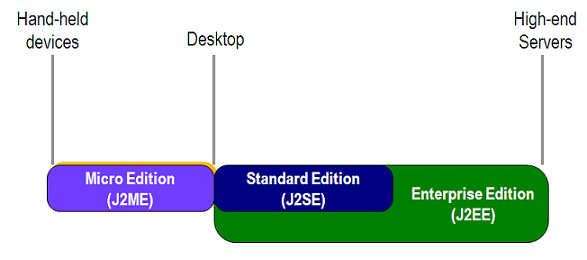
Java programming uses to produce byte codes and executes them. The first box indicates that the Java source code is located in a. Java file that is processed with a Java compiler called javac. The Java compiler produces a file called a. class file, which contains the byte code. The .Class file is then loaded across the network or loaded locally on your machine into the execution environment is the Java virtual machine, which interprets and executes the byte code.

#### Java Architecture

Java architecture provides a portable, robust, high performing environment for development. Java provides portability by compiling the byte codes for the Java Virtual Machine, which is then interpreted on each platform by the run-time environment. Java is a dynamic system, able to load code when needed from a machine in the same room or across the planet.

#### Compilation of code

When you compile the code, the Java compiler creates machine code (called byte code) for a hypothetical machine called Java Virtual Machine (JVM). The JVM is supposed to execute the byte code. The JVM is created for overcoming the issue of portability. The code is written and compiled for one machine and interpreted on all machines. This machine is called Java Virtual Machine.



**Compiling and interpreting Java Source Code**

During run-time the Java interpreter tricks the byte code file into thinking that it is running on a Java Virtual Machine. In reality this could be a Intel Pentium Windows 95 or SunSARC station running Solaris or Apple Macintosh running system and all could receive code from any computer through Internet and run the Applets.

Simple

Java was designed to be easy for the Professional programmer to learn and to use effectively. If you are an experienced C++ programmer, learning Java will be even easier. Because Java inherits the C/C++ syntax and many of the object oriented features of C++. Most of the confusing concepts from C++ are either left out of Java or implemented in a cleaner, more approachable manner. In Java there are a small number of clearly defined ways to accomplish a given task.

**Object-Oriented**

Java was not designed to be source-code compatible with any other language. This allowed the Java team the freedom to design with a blank slate. One outcome of this was a clean usable, pragmatic approach to objects. The object model in Java is simple and easy to extend, while simple types, such as integers, are kept as high-performance non-objects.

**Robust**

The multi-platform environment of the Web places extraordinary demands on a program, because the program must execute reliably in a variety of systems. The ability to create robust programs was given a high priority in the design of Java. Java is strictly typed language; it checks your code at compile time and run time.

Java virtually eliminates the problems of memory management and de-allocation, which is completely automatic. In a well-written Java program, all run time errors can –and should –be managed by your program.

**TECHNOLOGY INFRASTRUCTURE:**

**CORE JAVA:**

Java can be used to create two types of programs: application and applet. An application is a program that runs on your computer, under the operating system of that computer. That is, an application created by java is more or less like one created using C or C++. When used to create application, java is not much different from any other computer language. Rather, it is java’s ability to create applets that makes it important. An applet is an application designed to be transmitted over the internet and executed by a java-compatible Web Browser. An applet is actually a tiny java program, dynamically downloaded across the network, just like an image, sound file, or video clip. The important difference is that an applet is an intelligent program, not just an animation or media file. In other words, an applet is a program that can react to user input and dynamically change-not just run the same animation or sound over and over.

Java having a major roll in internet and the intranet application. The reason for this is quite simple: Java expends the universe pf objects that can move about freely in cyberspace. In a network, two very broad categories ob objects are transmitted between the server and your personal computer: passive information and dynamic, active programs.

**SECURITY**

As you are likely aware, every time that you download a “normal” program , you are risking viral infection. Prior to java, most users did not download executable programs frequently, and those who did scanned them for viruses prior to execution. Even so, most users still worried about the possibility of infecting their system with a virus. When you use a java-compatible web browser, you can safely download java applets without fear of viral infection or malicious intent. Java achieves this protection by confining a java program to the java execution environment and not allowing it access to other parts of computer.

**PORTABILITY:**

Many types of computers and operating systems are in use throughout the world-and many are connected to the internet. For program to be dynamically downloaded to all the various type of platforms connected to the Internet, some means of generating portable executable code is needed.

**BYTECODE:**

The key that allows java to solve both the security and the portability problems just described is that output of a java compiler is not executable code. Rather, it is BYTECODE. Byte code is a highly optimized set of instruction designed to be executed by the java run-time system, which is called the Java Virtual Machine (JVM). That is, in its standard form, the JVM is an interpreted code.

**SIMPLE:**

Java was designed to be easy for the professional programmer to learn and use effectively. Assuming that you have some programming experience, you will not find java hard to master. If we know the basic concept of object-oriented programming, learning java will be even easier.

**OBJECT-ORIENTED:**

Object-Oriented programming is the core of java. In fact, all java programs are object-oriented-this isn’t an option the way that it is in C++, for example. OOP is so integral to java that you must understand its basic principles before you can write even simple java programs.

**ABSTRACTION:**

The essential element of object-oriented programming is abstraction. Humans manage complexity through abstraction. For example, people do not think of a car as a set of ten of individual parts. They think of it as a well-defined object with its own unique behavior. So this ignore the details of how the engine, transmission, and braking systems work.

**THE THREE-OOP PRINCIPLE:**

1. **ENCAPSULATION:**

Encapsulation is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse. One way to think about encapsulation is as a protective wrapper that prevents the code and data from begin arbitrarily accessed by other code defined outside of the wrapper. Access to he code and data inside the wrappers is tightly controlled through a well-defined interface. To relate this to the real world, consider the automatic transmission on an automobile. It encapsulates hundreds of bits of information about our engine, such as how much you are accelerating, the pitch of the surface you are on, and the position of the shift lever.

**B)INHERITANCE:**

Inheritance is the process by which one object acquires the properties of another object. This is important because it supports the concept of hierarchical classification. As mentioned earlier, most knowledge is made manageable by hierarchical classification. Inheritance interacts with encapsulation as well. If a given class encapsulates some attributes, then subclass will have the same attributes plus any that it adds as part of its specialization. Java supports two type of inheritance. They are,

**1) SINGLE INHERITANCE:**

The derived class is inherited from one super class.

1. **MULTILEVEL INHERITANCE:**

This contains the hierarchical of classes.

**POLYMORPHISM:**

Polymorphism (from the Greek, meaning “many forms”) is a feature that allows one interface to be used for a general class of actions. More generally, the concept of polymorphism is often expressed by the phrase “one interface, multiple methods”. This means that is possible to design a generic interface to a group or related activities. This helps reduce complexity by allowing the same interface to be used to specify a general class of action. It is the compiler’s job to select the specific action as it applies to each situation.

**SWING:**

The Swing toolkit includes a rich set of components for building GUIs and adding interactivity to Java applications. Swing includes all the components you would expect from a modern toolkit: table controls, list controls, tree controls, buttons, and labels. Swing is far from a simple component toolkit, however. It includes rich undo support, a highly customizable text package, integrated internationalization and accessibility support. To truly leverage the cross-platform capabilities of the Java platform, Swing supports numerous look and feels, including the ability to create your own look and feel. The ability to create a custom look and feel is made easier with Synth, a look and feel specifically designed to be customized. Swing wouldn't be a component toolkit without the basic user interface primitives such as drag and drop, event handling, customizable painting, and window management.

Swing is part of the Java Foundation Classes (JFC). The JFC also include other features important to a GUI program, such as the ability to add rich graphics functionality and the ability to create a program that can work in different languages and by users with different input devices.

The Swing toolkit includes a rich array of components: from basic components, such as buttons and check boxes, to rich and complex components, such as tables and text. Even deceptively simple components, such as text fields, offer sophisticated functionality, such as formatted text input or password field behavior. There are file browsers and dialogs to suit most needs, and if not, customization is possible. If none of Swing's provided components are exactly what you need, you can leverage the basic Swing component functionality to create your own.

Swing components facilitate efficient graphical user interface (GUI) development. These components are a collection of light weight visual components. Swing components contain a replacement for the heavyweight AWT components as well as complex user-interface components such as trees and tables. Swing is a set of classes that provides more powerful and flexible components than are possible with the AWT. In addition to that the familiar components such as buttons, check box and labels swings supplies several exciting additions including tabbed panes, scroll panes, trees and tables. Even familiar components such as buttons have more capabilities in swing. For example a button may have both an image and text string associated with it. Also the image can be changed as the state of button changes. Unlike AWT components swing components are not implemented by platform specific code instead they are return entirely in JAVA and, therefore, are platform-independent. The term lightweight is used to describe such elements. The number of classes and interfaces in the swing packages is substantial. The Swing architecture is shown in the figure given blow:

Application Code

JFC

Swing

Java 2D

AWT

Drag & Drop

Accessibility

**THE SWING COMPONENT CLASSES:**

|  |  |
| --- | --- |
| Class | Description |
| Abstract Button | Abstract super class for Swing Buttons |
| Button Group | Encapsulates a mutually exclusive set of Buttons |
| ImageIcon | Encapsulates an Icon |
| JApplet | The Swing version of Applet |
| JButton | The Swing Push Button Class |
| JCheckBox | The Swing CheckBox class |
| JComboBOx | Encapsulates a combobox |
| JLabel | The swing version of a Label |
| JRadioButton | The Swing version of a RadioButton |
| JScrollPane | Encapsulates a scrollabel window |
| JTabbedPane | Encapsulates a Tabbed window |
| JTable | Encapsulates a Table-based control |
| JTextField | The swing version of a text-field |
| JTree | Encapsulates a Tree-based control |

**ADVANTAGES OF SWINGS:**

* Wide variety of Components
* Pluggable Look and Feel
* MVC Architecture
* Keystroke Handling
* Action Objects
* Nested containers
* Customized Dialogs
* Compound Borders
* Standard Dialog Classes
* Structured Table and Tree Components
* Powerful Text Manipulation
* Generic Undo Capabilities
* Accessibility Support

**JDBC Driver Types**

The JDBC drivers that we are aware of at this time fit into one of four categories:

* JDBC-ODBC bridge plus ODBC driver
* Native-API partly-Java driver
* JDBC-Net pure Java driver
* Native-protocol pure Java driver

**JDBC-ODBC Bridge**

If possible, use a Pure Java JDBC driver instead of the Bridge and an ODBC driver. This completely eliminates the client configuration required by ODBC. It also eliminates the potential that the Java VM could be corrupted by an error in the native code brought in by the Bridge (that is, the Bridge native library, the ODBC driver manager library, the ODBC driver library, and the database client library).

**What Is the JDBC- ODBC Bridge?**

The JDBC-ODBC Bridge is a JDBC driver, which implements JDBC operations by translating them into ODBC operations. To ODBC it appears as a normal application program. The Bridge implements JDBC for any database for which an ODBC driver is available. The Bridge is implemented as the

Sun.jdbc.odbc Java package and contains a native library used to access ODBC. The Bridge is a joint development of Innersole and Java Soft.

**JDBC connectivity**

The JDBC provides database-independent connectivity between the J2EE platform and a wide range of tabular data sources. JDBC technology allows an Application Component Provider to:

* Perform connection and authentication to a database server
* Manager transactions
* Move SQL statements to a database engine for preprocessing and execution
* Execute stored procedures
* Inspect and modify the results from Select statements

**Database:**

A database management system (DBMS) is computer software designed for the purpose of managing databases, a large set of structured data, and run operations on the data requested by numerous users. Typical examples of DBMSs include Oracle, DB2, Microsoft Access, Microsoft SQL Server, Firebird, PostgreSQL, MySQL, SQLite, FileMaker and Sybase Adaptive Server Enterprise. DBMSs are typically used by Database administrators in the creation of Database systems. Typical examples of DBMS use include accounting, human resources and customer support systems.

Originally found only in large companies with the computer hardware needed to support large data sets, DBMSs have more recently emerged as a fairly standard part of any company back office.

**Description**

A DBMS is a complex set of software programs that controls the organization, storage, management, and retrieval of data in a database. A DBMS includes:

* A modeling language to define the schema of each database hosted in the DBMS, according to the DBMS data model.
* The four most common types of organizations are the hierarchical, network, relational and object models. Inverted lists and other methods are also used. A given database management system may provide one or more of the four models. The optimal structure depends on the natural organization of the application's data, and on the application's requirements (which include transaction rate (speed), reliability, maintainability, scalability, and cost).
* The dominant model in use today is the ad hoc one embedded in SQL, despite the objections of purists who believe this model is a corruption of the relational model, since it violates several of its fundamental principles for the sake of practicality and performance. Many DBMSs also support the Open Database Connectivity API that supports a standard way for programmers to access the DBMS.
* Data structures (fields, records, files and objects) optimized to deal with very large amounts of data stored on a permanent data storage device (which implies relatively slow access compared to volatile main memory).
* A database query language and report writer to allow users to interactively interrogate the database, analyze its data and update it according to the users privileges on data.
* It also controls the security of the database.
* Data security prevents unauthorized users from viewing or updating the database. Using passwords, users are allowed access to the entire database or subsets of it called subschemas. For example, an employee database can contain all the data about an individual employee, but one group of users may be authorized to view only payroll data, while others are allowed access to only work history and medical data.
* If the DBMS provides a way to interactively enter and update the database, as well as interrogate it, this capability allows for managing personal databases. However, it may not leave an audit trail of actions or provide the kinds of controls necessary in a multi-user organization. These controls are only available when a set of application programs are customized for each data entry and updating function.
* A transaction mechanism, that ideally would guarantee the ACID properties, in order to ensure data integrity, despite concurrent user accesses (concurrency control), and faults (fault tolerance).
* It also maintains the integrity of the data in the database.
* The DBMS can maintain the integrity of the database by not allowing more than one user to update the same record at the same time. The DBMS can help prevent duplicate records via unique index constraints; for example, no two customers with the same customer numbers (key fields) can be entered into the database. See ACID properties for more information (Redundancy avoidance).

The DBMS accepts requests for data from the application program and instructs the operating system to transfer the appropriate data.

When a DBMS is used, information systems can be changed much more easily as the organization's information requirements change. New categories of data can be added to the database without disruption to the existing system.

Organizations may use one kind of DBMS for daily transaction processing and then move the detail onto another computer that uses another DBMS better suited for random inquiries and analysis. Overall systems design decisions are performed by data administrators and systems analysts. Detailed database design is performed by database administrators.

Database servers are specially designed computers that hold the actual databases and run only the DBMS and related software. Database servers are usually multiprocessor computers, with RAID disk arrays used for stable storage. Connected to one or more servers via a high-speed channel, hardware database accelerators are also used in large volume transaction processing environments.

DBMSs are found at the heart of most database applications. Sometimes DBMSs are built around a private multitasking kernel with built-in networking support although nowadays these functions are left to the operating system.

## Networking

This article is about a client/server multi-threaded socket class. The thread is optional since the developer is still responsible to decide if needs it. There are other Socket classes here and other places over the Internet but none of them can provide feedback (event detection) to your application like this one does. It provides you with the following events detection: connection established, connection dropped, connection failed and data reception (including 0 byte packet).

## Description

This article presents a new socket class which supports both TCP and UDP communication. But it provides some advantages compared to other classes that you may find here or on some other Socket Programming articles. First of all, this class doesn't have any limitation like the need to provide a window handle to be used. This limitation is bad if all you want is a simple console application. So this library doesn't have such a limitation. It also provides threading support automatically for you, which handles the socket connection and disconnection to a peer. It also features some options not yet found in any socket classes that I have seen so far. It supports both client and server sockets. A server socket can be referred as to a socket that can accept many connections. And a client socket is a socket that is connected to server socket. You may still use this class to communicate between two applications without establishing a connection. In the latter case, you will want to create two UDP server sockets (one for each application). This class also helps reduce coding need to create chat-like applications and IPC (Inter-Process Communication) between two or more applications (processes). Reliable communication between two peers is also supported with TCP/IP with error handling. You may want to use the smart addressing operation to control the destination of the data being transmitted (UDP only). TCP operation of this class deals only with communication between two peers

**Analysis of Network Client Server**

### TCP/IP stack

The TCP/IP stack is shorter than the OSI one:



TCP is a connection-oriented protocol; UDP (User Datagram Protocol) is a connectionless protocol.

IP datagram’s

The IP layer provides a connectionless and unreliable delivery system. It considers each datagram independently of the others. Any association between datagram must be supplied by the higher layers. The IP layer supplies a checksum that includes its own header. The header includes the source and destination addresses. The IP layer handles routing through an Internet. It is also responsible for breaking up large datagram into smaller ones for transmission and reassembling them at the other end.

### UDP

UDP is also connectionless and unreliable. What it adds to IP is a checksum for the contents of the datagram and port numbers. These are used to give a client/server model - see later.

### TCP

TCP supplies logic to give a reliable connection-oriented protocol above IP. It provides a virtual circuit that two processes can use to communicate.

### Internet addresses

In order to use a service, you must be able to find it. The Internet uses an address scheme for machines so that they can be located. The address is a 32 bit integer which gives the IP address. This encodes a network ID and more addressing. The network ID falls into various classes according to the size of the network address.

### 

### Network address

Class A uses 8 bits for the network address with 24 bits left over for other addressing. Class B uses 16 bit network addressing. Class C uses 24 bit network addressing and class D uses all 32.

### Subnet address

Internally, the UNIX network is divided into sub networks. Building 11 is currently on one sub network and uses 10-bit addressing, allowing 1024 different hosts.

### Host address

8 bits are finally used for host addresses within our subnet. This places a limit of 256 machines that can be on the subnet.

### Total address



The 32 bit address is usually written as 4 integers separated by dots.

### Port addresses

A service exists on a host, and is identified by its port. This is a 16 bit number. To send a message to a server, you send it to the port for that service of the host that it is running on. This is not location transparency! Certain of these ports are "well known".

### Sockets

A socket is a data structure maintained by the system to handle network connections. A socket is created using the call socket. It returns an integer that is like a file descriptor. In fact, under Windows, this handle can be used with ReadFile .

#include <sys/types.h>

#include <sys/socket.h>

int socket(int family, int type, int protocol);

Here "family" will be AF\_INET for IP communications, protocol will be zero, and type will depend on whether TCP or UDP is used. Two processes wishing to communicate over a network create a socket each. These are similar to two ends of a pipe - but the actual pipe does not yet exist.

Create a server socket that listens for a client to connect

socket(int af, int type, int protocol)

This method creates the socket

bind(SOCKET s, const struct sockaddr FAR \* name, int namelen)

Associates a local address with a socket This routine is used on an unconnected datagram or stream socket, before subsequent connects or listens. When a socket is created with socket, it exists in a name space (address family), but it has no name assigned. bind establishes the local association (host address/port number) of the socket by assigning a local name to an unnamed socket. In the Internet address family, a name consists of several components. For SOCK\_DGRAM and SOCK\_STREAM, the name consists of three parts: a host address, the protocol number (set implicitly to UDP or TCP, respectively), and a port number which identifies the application. If an application does not care what address is assigned to it, it may specify an Internet address equal to INADDR\_ANY, a port equal to 0, or both. If the Internet address is equal to INADDR\_ANY, any appropriate network interface will be used; this simplifies application programming in the presence of multi- homed hosts. If the port is specified as 0, the Windows Sockets implementation will assign a unique port to the application with a value between 1024 and 5000. The application may use getsockname after bind to learn the address that has been assigned to it, but note that getsockname will not necessarily fill in the Internet address until the socket is connected, since several Internet addresses may be valid if the host is multi-homed. If no error occurs, bind returns 0. Otherwise, it returns SOCKET\_ERROR, and a specific error code may be retrieved by calling WSAGetLastError.

listen(SOCKET s, int backlog )

Establishes a socket to listen to a incoming connection To accept connections, a socket is first created with socket, a backlog for incoming connections is specified with listen, and then the connections are accepted with accept. listen applies only to sockets that support connections, i.e. those of type SOCK\_STREAM. The socket s is put into "passive'' mode where incoming connections are acknowledged and queued pending acceptance by the process. This function is typically used by servers that could have more than one connection request at a time: if a connection request arrives with

the queue full, the client will receive an error with an indication of WSAECONNREFUSED. listen

attempts to continue to function rationally when there are no available descriptors. It will accept connections until the queue is emptied. If descriptors become available, a later call to listen or accept will re-fill the queue to the current or most recent "backlog'', if possible, and resume listening for incoming connections.

accept(SOCKET s, struct sockaddr FAR \* addr, int FAR \* addrlen)

This routine extracts the first connection on the queue of pending connections on s, creates a new socket with the same properties as s and returns a handle to the new socket. If no pending connections are present on the queue, and the socket is not marked as non- blocking, accept blocks the caller until a connection is present. If the socket is marked non-blocking and no pending connections are present on the queue, accept returns an error as described below. The accepted socket may not be used to accept more connections. The original socket remains open. The argument addr is a result parameter that is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the addr parameter is determined by the address family in which the communication is occurring.

The addrlen is a value-result parameter; it should initially contain the amount of space pointed to by addr; on return it will contain the actual length (in bytes) of the address returned. This call is used with connection-based socket types such as SOCK\_STREAM. If addr and/or addrlen are equal to NULL, then no information about the remote address of the accepted socket is returned.

closesocket(SOCKET s)

closes a socket

## Making client connection with server

In order to create a socket that connects to an other socket uses most of the functions from the previous code with the exception of a struct called HOSTENT

HOSTENT:

This struct is used to tell the This struct is used to tell the socket to which computer and port to connect to. These struct can appear as LPHOSTENT, but it actually means that they are pointer to HOSTENT.

### Client key function

Most of the functions that have been used for the client to connect to the server are the same as the server with the exception of a few. I will just go through the different functions that have been used for the client.

gethostbyname(const char\* FAR name)

gethostbyname returns a pointer to a hostent structure as described under gethostbyaddr. The contents of this structure correspond to the hostname name. The pointer which is returned points to a structure which is allocated by the Windows Sockets implementation. The application must never attempt to modify this structure or to free any of its components. Furthermore, only one copy of this structure is allocated per thread, and so the application should copy any information which it needs before issuing any other Windows Sockets API calls. A gethostbyname implementation must not resolve IP address strings passed to it. Such a request should be treated exactly as if an unknown host name were passed. An application with an IP address string to resolve should use inet\_addr to convert the string to an IP address, then gethostbyaddr to obtain the hostent structure.

### Part 2 - Send / recieve

Up to this point we have managed to connect with our client to the server. Clearly this is not going to be enough in a real-life application. In this section we are going to look into more details how to use the send/recv functions in order to get some communication going between the two applications.

Factually this is not going to be difficult because most of the hard work has been done setting up the server and the client app. before going into the code we are going to look into more details the two functions send(SOCKET s, const char FAR \* buf, int len, int flags) send is used on connected datagram or stream sockets and is used to write outgoing data on a socket. For datagram sockets, care must be taken not to exceed the maximum IP packet size of the underlying subnets, which is given by the iMaxUdpDg element in the WSAData structure returned by WSAStartup.

If the data is too long to pass atomically through the underlying protocol the error WSAEMSGSIZE is returned, and no data is transmitted.

recv(SOCKET s, const char FAR \* buf, int len, int flags)

For sockets of type SOCK\_STREAM, as much information as is currently available up to the size of the buffer supplied is returned. If the socket has been configured for in- line reception of out-of-band data (socket option SO\_OOBINLINE) and out-of-band data is unread, only out-of-band data will be returned. The application may use the ioctlsocket SIOCATMARK to determine whether any more out-of-band data remains to be read.

### part 3 - Read unknow size of data from client

Us mentioned earlier in part 2, we are noe going to expand on the way that we receive data. The problem we had before is that if we did not know the size of data that we where expecting, then the would end up with problems.

In order to fix this here we create a new function that receive a pointer to the client socket, and then read a char at the time, placing each char into a vector until we find the '\n' character that signifies the end of the message.

Thi@ÏÏð8@ö;‑early not a robust or industrial way the read data from one socket to an other, because but its a way to start reading unknown length strings. the function will be called after the accept method

**Proxy Server**

In computer networks, a **proxy server** is a server (a computer system or an application program) which services the requests of its clients by forwarding requests to other servers. A client connects to the proxy server, requesting some service, such as a file, connection, web page, or other resource, available from a different server. The proxy server provides the resource by connecting to the specified server and requesting the service on behalf of the client. A proxy server may optionally alter the client's request or the server's response, and sometimes it may serve the request without contacting the specified server. In this case, it would 'cache' the first request to the remote server, so it could save the information for later, and make everything as fast as possible.

A proxy server that passes all requests and replies unmodified is usually called a gateway or sometimes *tunneling proxy*.

A proxy server can be placed in the user's local computer or at various points between the user and the destination servers or the Internet.

## Types and functions

Proxy servers implement one or more of the following functions:

### Caching proxy server

A **caching proxy** server accelerates service requests by retrieving content saved from a previous request made by the same client or even other clients. Caching proxies keep local copies of frequently requested resources, allowing large organizations to significantly reduce their upstream bandwidth usage and cost, while significantly increasing performance. Most ISPs and large businesses have a caching proxy. These machines are built to deliver superb file system performance (often with RAID and journaling) and also contain hot-rodded versions of TCP. Caching proxies were the first kind of proxy server.

The HTTP 1.0 and later protocols contain many types of headers for declaring static (cacheable) content and verifying content freshness with an original server, e.g. ETAG (validation tags), If-Modified-Since (date-based validation), Expiry (timeout-based invalidation), etc. Other protocols such as DNS support expiry only and contain no support for validation.

Some poorly-implemented caching proxies have had downsides (e.g., an inability to use user authentication). Some problems are described in RFC 3143 (Known HTTP Proxy/Caching Problems).

### 

### Web proxy

A proxy that focuses on WWW traffic is called a "web proxy". The most common use of a web proxy is to serve as a web cache. Most proxy programs (e.g. Squid) provide a means to deny access to certain URLs in a blacklist, thus providing content filtering. This is usually used in a corporate environment, though with the increasing use of Linux in small businesses and homes, this function is no longer confined to large corporations. Some web proxies reformat web pages for a specific purpose or audience (e.g., cell phones and PDAs).

AOL dialup customers used to have their requests routed through an extensible proxy that 'thinned' or reduced the detail in JPEG pictures. This sped up performance, but caused trouble, either when more resolution was needed or when the thinning program produced incorrect results. This is why in the early days of the Internet many web pages would contain a link saying "AOL Users Click Here" to bypass the web proxy and to avoid the bugs in the thinning software.

### Content-filtering web proxy

A content-filtering web proxy server provides administrative control over the content that may be relayed through the proxy. It is commonly used in commercial and non-commercial organizations (especially schools) to ensure that Internet usage conforms to acceptable use policy.

Common methods used for content filtering include: URL or DNS blacklists, URL regex filtering, MIME filtering, or content keyword filtering. Some products have been known to employ content analysis techniques to look for traits commonly used by certain types of content providers.

A content filtering proxy will often support user authentication, to control web access. It also usually produces logs, either to give detailed information about the URLs accessed by specific users, or to monitor bandwidth usage statistics. It may also communicate to daemon based and/or ICAP based antivirus software to provide security against virus and other malware by scanning incoming content in real time before it enters the network.

### Anonym zing proxy server

An anonymous proxy server (sometimes called a web proxy) generally attempts to anonymize web surfing. These can easily be overridden by site administrators, and thus rendered useless in some cases. There are different varieties of anonymizers.

**Access control**: Some proxy servers implement a logon requirement. In large organizations, authorized users must log on to gain access to the web. The organization can thereby track usage to individuals.

### 

### Hostile proxy

Proxies can also be installed by online criminals, in order to eavesdrop upon the dataflow between the client machine and the web. All accessed pages, as well as all forms submitted, can be captured and analyzed by the proxy operator. For this reason, passwords to online services (such as webmail and banking) should always be exchanged over a cryptographically secured connection, such as TLS.

### Intercepting proxy server

An **intercepting proxy** (also known as a "transparent proxy") combines a proxy server with a gateway. Connections made by client browsers through the gateway are redirected through the proxy without client-side configuration (or often knowledge).

Intercepting proxies are commonly used in businesses to prevent avoidance of acceptable use policy, and to ease administrative burden, since no client browser configuration is required.

It is often possible to detect the use of an intercepting proxy server by comparing the external IP address to the address seen by an external web server, or by examining the HTTP headers on the server side.

### Transparent and non-transparent proxy server

The term "transparent proxy" is most often used incorrectly to mean "intercepting proxy" (because the client does not need to configure a proxy and cannot directly detect that its requests are being proxied). Transparent proxies can be implemented using Cisco's WCCP (Web Cache Control Protocol). This proprietary protocol resides on the router and is configured from the cache, allowing the cache to determine what ports and traffic is sent to it via transparent redirection from the router. This redirection can occur in one of two ways: GRE Tunneling (OSI Layer 3) or MAC rewrites (OSI Layer 2).

However, RFC 2616 (Hypertext Transfer Protocol -- HTTP/1.1) offers different definitions:

"A 'transparent proxy' is a proxy that does not modify the request or response beyond what is required for proxy authentication and identification".

"A 'non-transparent proxy' is a proxy that modifies the request or response in order to provide some added service to the user agent, such as group annotation services, media type transformation, protocol reduction, or anonymity filtering".

### Forced proxy

The term "forced proxy" is ambiguous. It means both "intercepting proxy" (because it filters all traffic on the only available gateway to the Internet) and its exact opposite, "non-intercepting proxy" (because the user is forced to configure a proxy in order to access the Internet).

Forced proxy operation is sometimes necessary due to issues with the interception of TCP connections and HTTP. For instance interception of HTTP requests can affect the usability of a proxy cache, and can greatly affect certain authentication mechanisms. This is primarily because the client thinks it is talking to a server, and so request headers required by a proxy are unable to be distinguished from headers that may be required by an upstream server (esp authorization headers). Also the HTTP specification prohibits caching of responses where the request contained an authorization header.

**5. Other Nonfunctional Requirements**

5.1 Security Requirements

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**Eclipse IDE**

Eclipse is an open-source software framework written primarily in Java. In its default form it is an Integrated Development Environment (IDE) for Java developers, consisting of the Java Development Tools (JDT) and the Eclipse Compiler for Java (ECJ). Users can extend its capabilities by installing plug-ins written for the Eclipse software framework, such as development toolkits for other programming languages, and can write and contribute their own plug-in modules. Language packs are available for over a dozen languages.

**Architecture**

The basis for Eclipse is the Rich Client Platform (RCP). The following components constitute the rich client platform:

* OSGi - a standard bundling framework
* Core platform - boot Eclipse, run plug-ins
* the Standard Widget Toolkit (SWT) - a portable widget toolkit
* JFace - viewer classes to bring model view controller programming to SWT, file buffers, text handling, text editors
* the Eclipse Workbench - views, editors, perspectives, wizards

Eclipse's widgets are implemented by a widget toolkit for Java called SWT, unlike most Java applications, which use the Java standard Abstract Window Toolkit (AWT) or Swing. Eclipse's user interface also leverages an intermediate GUI layer called JFace, which simplifies the construction of applications based on SWT.

Eclipse employs plug-ins in order to provide all of its functionality on top of (and including) the rich client platform, in contrast to some other applications where functionality is typically hard coded. This plug-in mechanism is a lightweight software componentry framework. In addition to allowing Eclipse to be extended using other programming languages such as C and Python, the plug-in framework allows Eclipse to work with typesetting languages like LaTeX, networking applications such as telnet, and database management systems. The plug-in architecture supports writing any desired extension to the environment, such as for configuration management. Java and CVS support is provided in the Eclipse SDK.

The key to the seamless integration of tools with Eclipse is the plugin. With the exception of a small run-time kernel, everything in Eclipse is a plug-in. This means that a plug-in you develop integrates with Eclipse in exactly the same way as other plug-ins; in this respect, all features are created equal.

The Eclipse SDK includes the Eclipse Java Development Tools, offering an IDE with a built-in incremental Java compiler and a full model of the Java source files. This allows for advanced refactoring techniques and code analysis. The IDE also makes use of a workspace, in this case a set of metadata over a flat filespace allowing external file modifications as long as the corresponding workspace "resource" is refreshed afterwards. The Visual Editor project allows interfaces to be created interactively, hence allowing Eclipse to be used as a RAD tool.

The following is a list of notable projects and plugins for the Eclipse IDE.

These projects are maintained by the Eclipse community and hosted by the Eclipse Foundation.

1. **Core projects**

Rich Client Platform (Platform) is the core framework that all other Eclipse projects are built on.

Java Development Tools (JDT) provides support for core Java SE. This includes a standalone fast incremental compiler.

**Tools projects**

C/C++ Development Tools (CDT) adds support for C/C++ syntax highlighting, code formatting, debugger integration and project structures. Unlike the JDT project, the CDT project does not add a compiler and relies on an external tool chain.

Graphical Editing Framework (GEF) allows developers to build standalone graphical tools. Example use include circuit diagram design tools, activity diagram editors and WYSIWYG document editors.

**Web projects**

J2EE Standard Tools (JST) extends the core JDT to include support for Java EE projects. This includes EJBs, JSPs and Servlets.

PHP Development Tools (PDT)

Web Standard Tools (WST) adds standards compliant web development tools. These tools include editors for XML, HTML and CSS.

**Other projects**

Test and Performance Tools Platform (TPTP) which provides a platform that allows software developers to build test and performance tools, such as debuggers, profilers and benchmarking applications.

Business Intelligence and Reporting Tools Project (BIRT), an Eclipse-based open source reporting system for web applications, especially those based on Java

**SYSTEM DESIGN**

Design is multi-step process that focuses on data structure software architecture, procedural details, (algorithms etc.) and interface between modules. The design process also translates the requirements into the presentation of software that can be accessed for quality before coding begins.

Computer software design changes continuously as new methods; better analysis and broader understanding evolved. Software Design is at relatively early stage in its revolution.

Therefore, Software Design methodology lacks the depth, flexibility and quantitative nature that are normally associated with more classical engineering disciplines. However techniques for software designs do exist, criteria for design qualities are available and design notation can be applied.

**Module description:**

###### **Group Generation:**

In this module, all the nodes that are attached to the particular node will be displayed in the list and the left and the right node to which the message has to be sent are selected from the list. When we want to send a message the data will be sent to those nodes which are selected from the list.

###### **Path Discovery**

The shortest path from the source node i.e. the root to the left and the right is found out and the request is sent from the left and right nodes to the root node by which the shortest path is found and the message is sent along the shortest path found through the found shortest path.

**Recursive UniCast Trees**

* The MFT has no destination entry. Data received by a branching router Hb, has unicast destination address set to Hb (in REUNITE, data are addressed to MFT(s).dst). This choice makes the tree structure more stable than in REUNITE.
* Non branching routers simply keep group information in their MCT, whereas branching nodes keep MFT entries which are

Used to recursively create packet copies to reach all group members.

* Multicasting the data from the source node to the REUNITE (Recursive UNIcast Trees) server.
* Key idea of REUNITE is to separate multicast routing information in two tables: a multicast control table (MCT), which is stored in the control plane, and a multicast forwarding table (MFT), which is installed in the data plane.

**Hop By Hop Transmission**

* The HBH multicast protocol has a tree construction algorithm which finds the end-host in the specified subnet by checking node by node.
* This is achieved by checking the every node IP address against destination IP

**UML DIAGRAMS**

**Unified Modeling Language**:

The Unified Modeling Language allows the software engineer to express an analysis model using the modeling notation that is governed by a set of syntactic semantic and pragmatic rules.

A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagram, which is as follows.

* + User Model View
    1. This view represents the system from the users perspective.
    2. The analysis representation describes a usage scenario from the end-users perspective.
  + Structural model view
    1. In this model the data and functionality are arrived from inside the system.
    2. This model view models the static structures.
* Behavioral Model View

It represents the dynamic of behavioral as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.

* Implementation Model View

In this the structural and behavioral as parts of the system are represented as they are to be built.

* Environmental Model View

In this the structural and behavioral aspects of the environment in which the system is to be implemented are represented.

UML is specifically constructed through two different domains they are:

* UML Analysis modeling, this focuses on the user model and structural model views of the system.
* UML design modeling, which focuses on the behavioral modeling, implementation modeling and environmental model views.

Use case Diagrams represent the functionality of the system from a user’s point of view. Use cases are used during requirements elicitation and analysis to represent the functionality of the system. Use cases focus on the behavior of the system from external point of view.

Actors are external entities that interact with the system. Examples of actors include users like administrator, bank customer …etc., or another system like central database.

**UML Diagram**

**Use Case Diagram**

**SYSTEM TESTING AND MAINTENANCE**

**Testing**

Software Testing is the process used to help identify the correctness, completeness, security, and quality of developed computer software. Testing is a process of technical investigation, performed on behalf of stakeholders, that is intended to reveal quality-related information about the product with respect to the context in which it is intended to operate. This includes, but is not limited to, the process of executing a program or application with the intent of finding errors. Quality is not an absolute; it is value to some person. With that in mind, testing can never completely establish the correctness of arbitrary computer software; testing furnishes a criticism or comparison that compares the state and behavior of the product against a specification. An important point is that software testing should be distinguished from the separate discipline of Software Quality Assurance (SQA), which encompasses all business process areas, not just testing.

There are many approaches to software testing, but effective testing of complex products is essentially a process of investigation, not merely a matter of creating and following routine procedure. One definition of testing is "the process of questioning a product in order to evaluate it", where the "questions" are operations the tester attempts to execute with the product, and the product answers with its behavior in reaction to the probing of the tester[citation needed]. Although most of the intellectual processes of testing are nearly identical to that of review or inspection, the word testing is connoted to mean the dynamic analysis of the product—putting the product through its paces. Some of the common quality attributes include capability, reliability, efficiency, portability, maintainability, compatibility and usability. A good test is sometimes described as one which reveals an error; however, more recent thinking suggests that a good test is one which reveals information of interest to someone who matters within the project community.

Introduction

In general, software engineers distinguish software faults from software failures. In case of a failure, the software does not do what the user expects. A fault is a programming error that may or may not actually manifest as a failure. A fault can also be described as an error in the correctness of the semantic of a computer program. A fault will become a failure if the exact computation conditions are met, one of them being that the faulty portion of computer software executes on the CPU. A fault can also turn into a failure when the software is ported to a different hardware platform or a different compiler, or when the software gets extended. Software testing is the technical investigation of the product under test to provide stakeholders with quality related information.

Software testing may be viewed as a sub-field of Software Quality Assurance but typically exists independently (and there may be no SQA areas in some companies). In SQA, software process specialists and auditors take a broader view on software and its development. They examine and change the software engineering process itself to reduce the amount of faults that end up in the code or deliver faster.

Regardless of the methods used or level of formality involved the desired result of testing is a level of confidence in the software so that the organization is confident that the software has an acceptable defect rate. What constitutes an acceptable defect rate depends on the nature of the software. An arcade video game designed to simulate flying an airplane would presumably have a much higher tolerance for defects than software used to control an actual airliner.

A problem with software testing is that the number of defects in a software product can be very large, and the number of configurations of the product larger still. Bugs that occur infrequently are difficult to find in testing. A rule of thumb is that a system that is expected to function without faults for a certain length of time must have already been tested for at least that length of time. This has severe consequences for projects to write long-lived reliable software.

A common practice of software testing is that it is performed by an independent group of testers after the functionality is developed but before it is shipped to the customer. This practice often results in the testing phase being used as project buffer to compensate for project delays. Another practice is to start software testing at the same moment the project starts and it is a continuous process until the project finishes.

Another common practice is for test suites to be developed during technical support escalation procedures. Such tests are then maintained in regression testing suites to ensure that future updates to the software don't repeat any of the known mistakes.

It is commonly believed that the earlier a defect is found the cheaper it is to fix it.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Time Detected | | | | |
| Time Introduced | Requirements | Architecture | Construction | System Test | Post-Release |
| Requirements | 1 | 3 | 5-10 | 10 | 10-100 |
| Architecture | - | 1 | 10 | 15 | 25-100 |
| Construction | - | - | 1 | 10 | 10-25 |

In counterpoint, some emerging software disciplines such as extreme programming and the agile software development movement, adhere to a "test-driven software development" model. In this process unit tests are written first, by the programmers (often with pair programming in the extreme programming methodology). Of course these tests fail initially; as they are expected to. Then as code is written it passes incrementally larger portions of the test suites. The test suites are continuously updated as new failure conditions and corner cases are discovered, and they are integrated with any regression tests that are developed.

Unit tests are maintained along with the rest of the software source code and generally integrated into the build process (with inherently interactive tests being relegated to a partially manual build acceptance process).

The software, tools, samples of data input and output, and configurations are all referred to collectively as a test harness.

## History

The separation of debugging from testing was initially introduced by Glenford J. Myers in his 1978 book the "Art of Software Testing". Although his attention was on breakage testing it illustrated the desire of the software engineering community to separate fundamental development activities, such as debugging, from that of verification. Drs. Dave Gelperin and William C. Hetzel classified in 1988 the phases and goals in software testing as follows: until 1956 it was the debugging oriented period, where testing was often associated to debugging: there was no clear difference between testing and debugging. From 1957-1978 there was the demonstration oriented period where debugging and testing was distinguished now - in this period it was shown, that software satisfies the requirements. The time between 1979-1982 is announced as the destruction oriented period, where the goal was to find errors. 1983-1987 is classified as the evaluation oriented period: intention here is that during the software lifecycle a product evaluation is provided and measuring quality. From 1988 on it was seen as prevention oriented period where tests were to demonstrate that software satisfies its specification, to detect faults and to prevent faults. Dr. Gelperin chaired the IEEE 829-1988 (Test Documentation Standard) with Dr. Hetzel writing the book "The Complete Guide of Software Testing". Both works were pivotal in to today's testing culture and remain a consistent source of reference. Dr. Gelperin and Jerry E. Durant also went on to develop High Impact Inspection Technology that builds upon traditional Inspections but utilizes a test driven additive.

## White-box and black-box testing

To meet Wikipedia's quality standards, this section may require cleanup.  
Please discuss this issue on the talk page, and/or replace this tag with a more specific message.

White box and black box testing are terms used to describe the point of view a test engineer takes when designing test cases. Black box being an external view of the test object and white box being an internal view. Software testing is partly intuitive, but largely systematic. Good testing involves much more than just running the program a few times to see whether it works. Thorough analysis of the program under test, backed by a broad knowledge of testing techniques and tools are prerequisites to systematic testing. Software Testing is the process of executing software in a controlled manner; in order to answer the question “Does this software behave as specified?” Software testing is used in association with Verification and Validation. Verification is the checking of or testing of items, including software, for conformance and consistency with an associated specification. Software testing is just one kind of verification, which also uses techniques as reviews, inspections, walk-through. Validation is the process of checking what has been specified is what the user actually wanted.

* Validation: Are we doing the right job?
* Verification: Are we doing the job right?

In order to achieve consistency in the Testing style, it is imperative to have and follow a set of testing principles. This enhances the efficiency of testing within SQA team members and thus contributes to increased productivity. The purpose of this document is to provide overview of the testing, plus the techniques.

At SDEI, 3 levels of software testing is done at various SDLC phases

* Unit Testing: in which each unit (basic component) of the software is tested to verify that the detailed design for the unit has been correctly implemented
* Integration testing: in which progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a whole.
* System testing: in which the software is integrated to the overall product and tested to show that all requirements are met

A further level of testing is also done, in accordance with requirements:

* Acceptance testing: upon which the acceptance of the complete software is based. The clients often do this.
* Regression testing: is used to refer the repetition of the earlier successful tests to ensure that changes made in the software have not introduced new bugs/side effects.

In recent years the term grey box testing has come into common usage. The typical grey box tester is permitted to set up or manipulate the testing environment, like seeding a database, and can view the state of the product after his actions, like performing a SQL query on the database to be certain of the values of columns. It is used almost exclusively of client-server testers or others who use a database as a repository of information, but can also apply to a tester who has to manipulate XML files (DTD or an actual XML file) or configuration files directly. It can also be used of testers who know the internal workings or algorithm of the software under test and can write tests specifically for the anticipated results. For example, testing a data warehouse implementation involves loading the target database with information, and verifying the correctness of data population and loading of data into the correct tables.

## Test levels

* Unit testing tests the minimal software component and sub-component or modules by the programmers.
* Integration testing exposes defects in the interfaces and interaction between integrated components (modules).
* Functional testing tests the product according to programmable work.
* System testing tests an integrated system to verify/validate that it meets its requirements.
* Acceptance testing testing can be conducted by the client. It allows the end-user or customer or client to decide whether or not to accept the product. Acceptance testing may be performed after the testing and before the implementation phase. See also Development stage
  + Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing, before the software goes to beta testing.
  + Beta testing comes after alpha testing. Versions of the software, known as beta versions, are released to a limited audience outside of the company. The software is released to groups of people so that further testing can ensure the product has few faults or bugs. Sometimes, beta versions are made available to the open public to increase the feedback field to a maximal number of future users.

It should be noted that although both Alpha and Beta are referred to as testing it is in fact use emersion. The rigors that are applied are often unsystematic and many of the basic tenets of testing process are not used. The Alpha and Beta period provides insight into environmental and utilization conditions that can impact the software.

After modifying software, either for a change in functionality or to fix defects, a regression test re-runs previously passing tests on the modified software to ensure that the modifications haven't unintentionally caused a regression of previous functionality. Regression testing can be performed at any or all of the above test levels. These regression tests are often automated.

## Test cases, suites, scripts and scenarios

A test case is a software testing document, which consists of event, action, input, output, expected result and actual result. Clinically defined (IEEE 829-1998) a test case is an input and an expected result. This can be as pragmatic as 'for condition x your derived result is y', whereas other test cases described in more detail the input scenario and what results might be expected. It can occasionally be a series of steps (but often steps are contained in a separate test procedure that can be exercised against multiple test cases, as a matter of economy) but with one expected result or expected outcome. The optional fields are a test case ID, test step or order of execution number, related requirement(s), depth, test category, author, and check boxes for whether the test is automatable and has been automated. Larger test cases may also contain prerequisite states or steps, and descriptions. A test case should also contain a place for the actual result. These steps can be stored in a word processor document, spreadsheet, database or other common repository. In a database system, you may also be able to see past test results and who generated the results and the system configuration used to generate those results. These past results would usually be stored in a separate table.

The term test script is the combination of a test case, test procedure and test data. Initially the term was derived from the byproduct of work created by automated regression test tools. Today, test scripts can be manual, automated or a combination of both.

The most common term for a collection of test cases is a test suite. The test suite often also contains more detailed instructions or goals for each collection of test cases. It definitely contains a section where the tester identifies the system configuration used during testing. A group of test cases may also contain prerequisite states or steps, and descriptions of the following tests.

Collections of test cases are sometimes incorrectly termed a test plan. They might correctly be called a test specification. If sequence is specified, it can be called a test script, scenario or procedure.

## A sample testing cycle

Although testing varies between organizations, there is a cycle to testing:

1. Requirements Analysis: Testing should begin in the requirements phase of the software development life cycle.

During the design phase, testers work with developers in determining what aspects of a design are testable and under what parameter those tests work.

1. Test Planning: Test Strategy, Test Plan(s), Test Bed creation.
2. Test Development: Test Procedures, Test Scenarios, Test Cases, Test Scripts to use in testing software.
3. Test Execution: Testers execute the software based on the plans and tests and report any errors found to the development team.
4. Test Reporting: Once testing is completed, testers generate metrics and make final reports on their test effort and whether or not the software tested is ready for release.
5. Retesting the Defects

Not all errors or defects reported must be fixed by a software development team. Some may be caused by errors in configuring the test software to match the development or production environment. Some defects can be handled by a workaround in the production environment. Others might be deferred to future releases of the software, or the deficiency might be accepted by the business user. There are yet other defects that may be rejected by the development team (of course, with due reason) if they deem it inappropriate to be called a defect.

SYSTEM IMPLEMENTATION

Implementation is the most crucial stage in achieving a successful system and giving the user’s confidence that the new system is workable and effective. Implementation of a modified application to replace an existing one. This type of conversation is relatively easy to handle, provide there are no major changes in the system.

Each program is tested individually at the time of development using the data and has verified that this program linked together in the way specified in the programs specification, the computer system and its environment is tested to the satisfaction of the user. The system that has been developed is accepted and proved to be satisfactory for the user. And so the system is going to be implemented very soon. A simple operating procedure is included so that the user can understand the different functions clearly and quickly.

Initially as a first step the executable form of the application is to be created and loaded in the common server machine which is accessible to all the user and the server is to be connected to a network. The final stage is to document the entire system which provides components and the operating procedures of the system.

SCOPE FOR FUTURE DEVELOPMENT

Every application has its own merits and demerits. The project has covered almost all the requirements. Further requirements and improvements can easily be done since the coding is mainly structured or modular in nature. Changing the existing modules or adding new modules can append improvements. Further enhancements can be made to the application, so that the web site functions very attractive and useful manner than the present one.

CONCLUSION

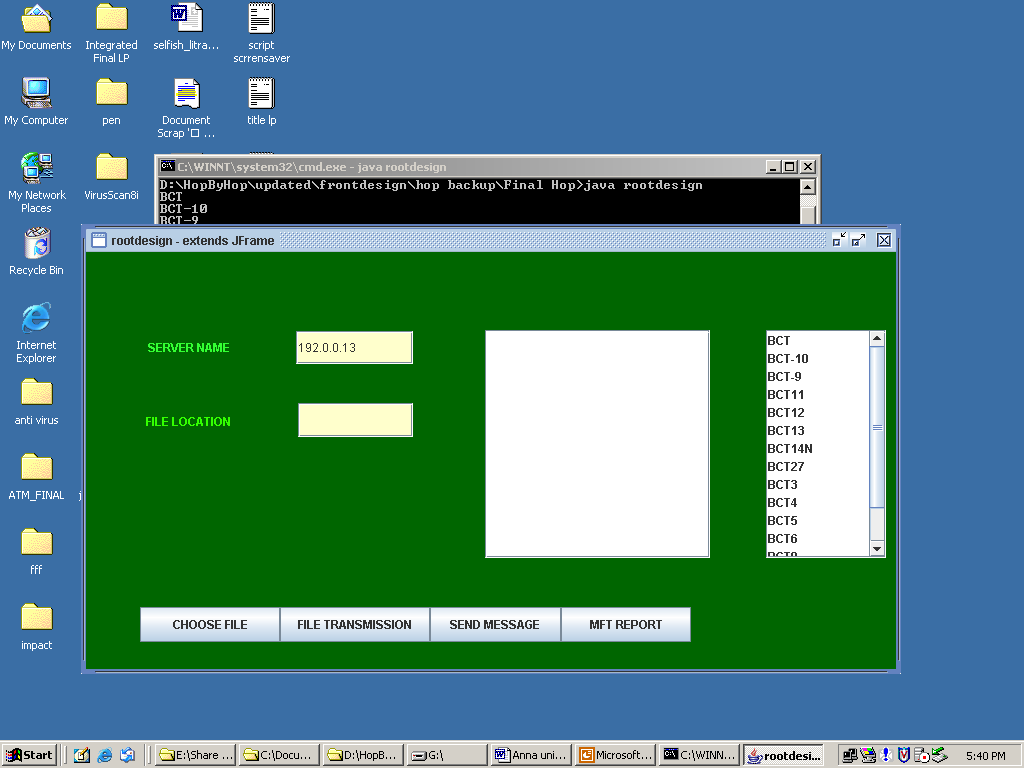
In this paper, we have studied the problem of combating Internet worms. To that end, we have developed a branching process model to characterize the propagation of Internet worms. Unlike deterministic epidemic models studied in the literature, this model allows us to characterize the early phase of worm propagation. Using the branching process model, we are able to provide a precise bound M on the total number of scans that ensure that the worm will eventually die out. Further, from our model, we also obtain the probability that the total number of hosts that the worm infects is below a certain level, as a function of the scan limit. The insights gained from analyzing this model also allow us to develop an effective and automatic worm containment strategy that does not let the worm propagate beyond the early stages of infection. Our strategy can effectively contain both fast scan worms and slow scan worms without knowing the worm signature in advance or needing to explicitly detect the worm. We show via simulations and real trace data that the containment strategy is both effective and non-intrusive.

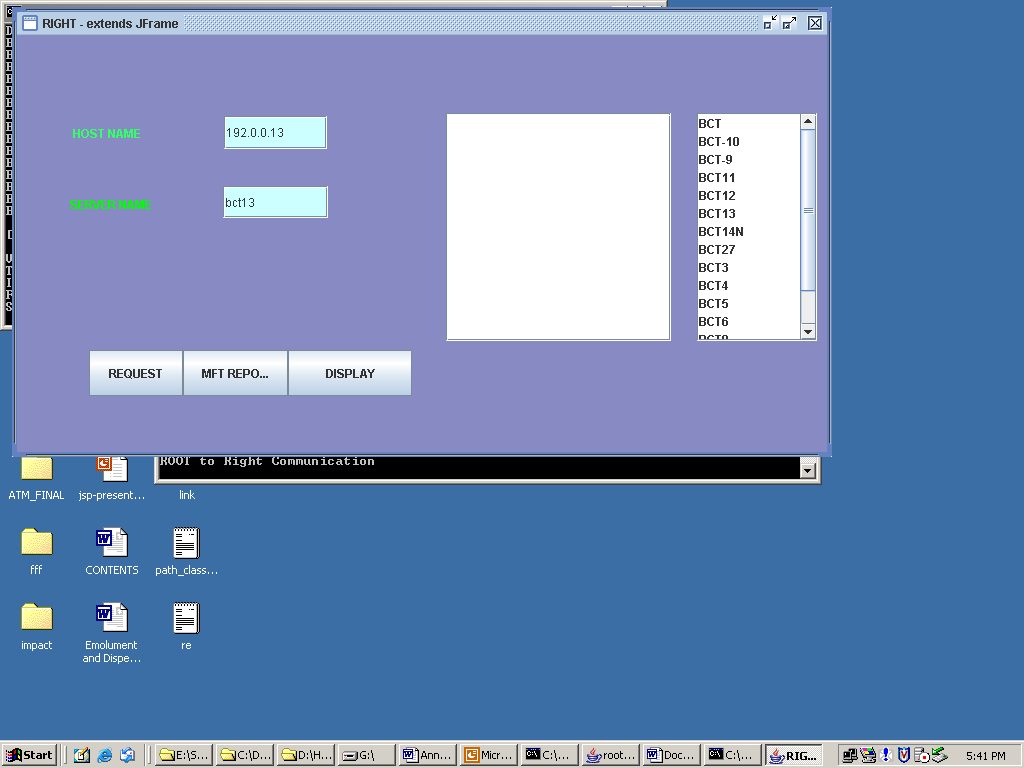
BIBLIOGRAPHY

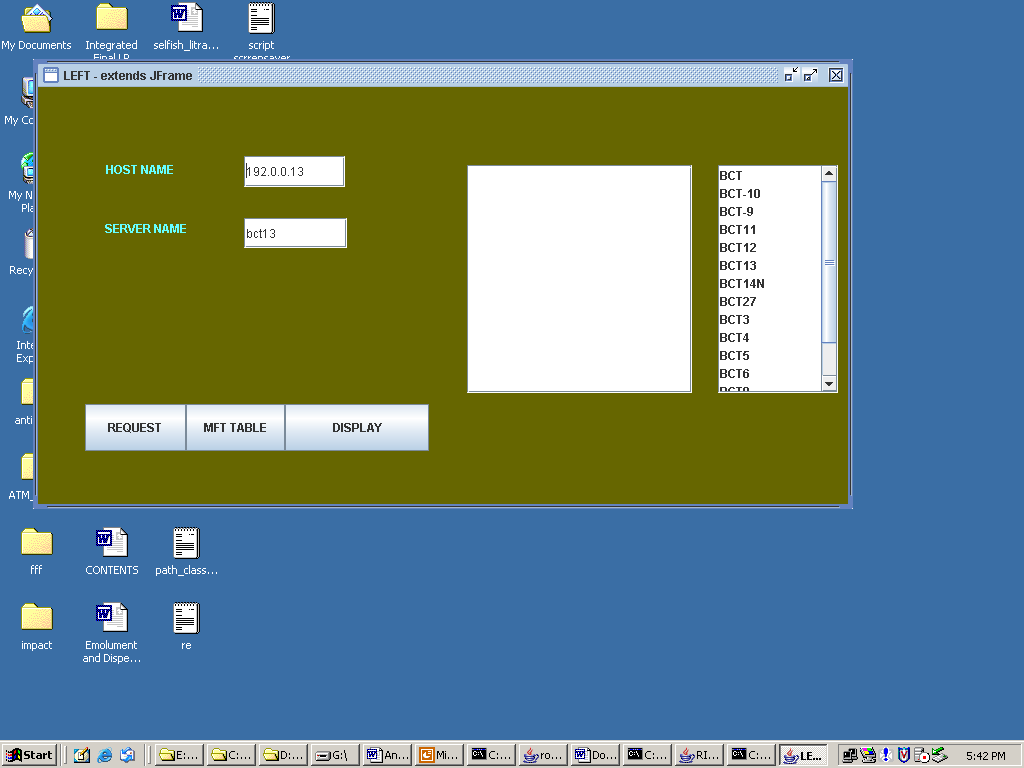
|  |  |
| --- | --- |
| Core Java™ 2 Volume I – Fundamentals 7th Edition | * Cay S. Hortsman |
| Pearson Education – Sun Microsystems | Gary Cornell |
|  |  |
| Core Java™ 2 Volume II – Advanced | * Cay S. Hortsman |
| Pearson Education – Sun Microsystems | Gary Cornell |
|  |  |
|  |  |
|  |  |
|  |  |
| JAVA 2: The Complete Reference | * Hert Schildt, |
| SPD | Herbert Schildt |
|  |  |
| Effective Java – Programming Language Guide | * Joshua Bloch |
| Pearson Education – Sun Microsystems |  |
|  |  |
| Java Networking Best Practices | * George Reese |
| O’Reilly – SPD |  |
|  |  |
| Pure JFC Swing | * Sathyaraj Pantham |
| O’Reilly – SPD | Sam Griffith |

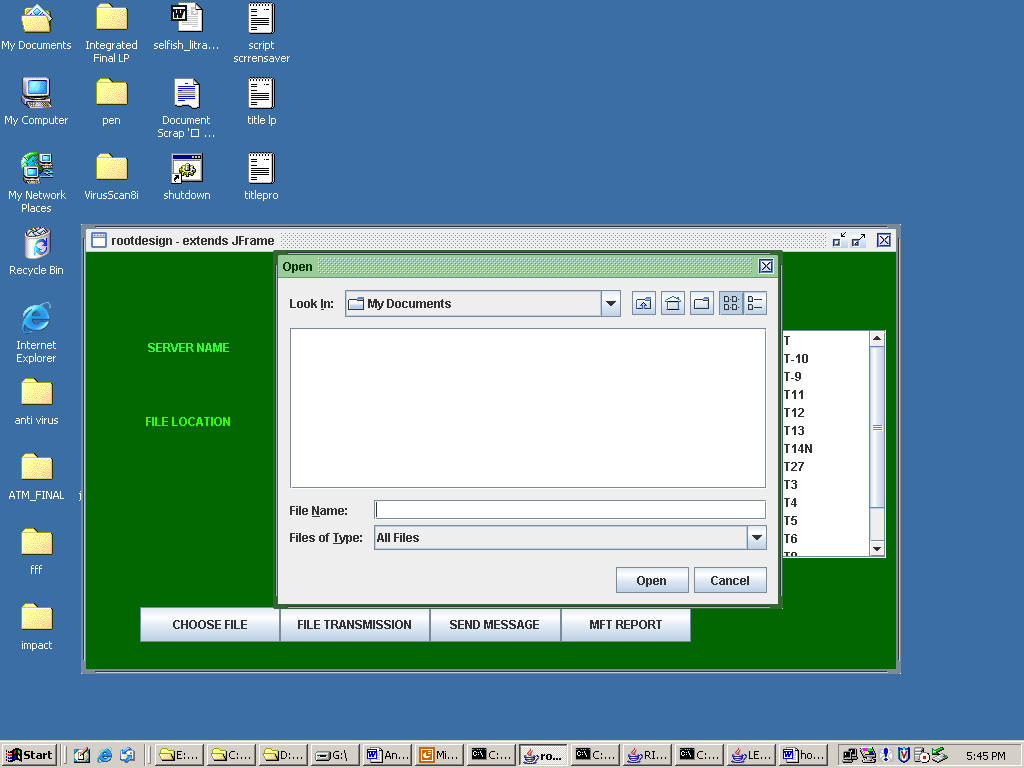
**APPENDIX**

**SCREEN SHOTS**









**Sample Code:**

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

import java.net.\*;

import java.io.\*;

import java.util.\*;

import java.sql.\*;

class mains extends Thread{

public static ServerSocket serverSocket1;

public static DataOutputStream out1=null;

public static DataInputStream in1=null;

public static String flag;

Socket right=null;

int port;

int d1,d2;

public static String location;

Connection con;

Statement st;

ResultSet rs;

//String rlocation;

//String rhost;

String llocation;

String lipaddress;

String rrlocation;

String rrrequest;

String rrhost;

String root;

public void run()

{

try{

serverSocket1 = new ServerSocket(1039); //this is to right and left (indirect path but it is very less in weight)

System.out.println("ROOT to Right Communication");

java.util.Timer myTimer = new java.util.Timer();

//delay d=new delay();

//java.util.Timer time=new java.util.Timer();

//time.schedule(d,2000);

while(true)

{

right=serverSocket1.accept();

System.out.println("Connected to Left and Right child Node");

out1=new DataOutputStream(right.getOutputStream());

in1=new DataInputStream(right.getInputStream());

if(right.isConnected())

{

flag="right";

System.out.println("Connected From Right Node...");

// Thread.sleep();

// mainServer.location="right";

String req=mains.in1.readUTF(); // ist read from right

if(req.equals("rightrequest"))

{

rrlocation=mains.in1.readUTF(); //2nd

rrhost=mains.in1.readUTF(); //3rd

// Thread.sleep();

root=InetAddress.getLocalHost().toString();

System.out.println("Root received the data from Right");

JOptionPane.showMessageDialog(null," Right Node Requested For Multicast Comminication ","Request From Right",JOptionPane.OK\_OPTION);

}

else

{

llocation=mains.in1.readUTF(); //2

lipaddress=mains.in1.readUTF(); //3

rrrequest=mains.in1.readUTF(); //4

//Thread.sleep();

rrlocation=mains.in1.readUTF(); //5

rrhost=mains.in1.readUTF(); //6

root=InetAddress.getLocalHost().toString();

System.out.println("Root received the data from Left and Right");

JOptionPane.showMessageDialog(null,"Left Node Requested For Multicast Comminication ","Request From Left",JOptionPane.OK\_OPTION);

}

db();

PreparedStatement ps = con.prepareStatement("insert into MFT values(?,?,?)");

ps.setString(1,rrlocation);

ps.setString(2,rrhost);

ps.setString(3,root);

ps.executeUpdate();

System.out.println("1 Row Updated through mainerver class");

con.close();

task1 myTask1 = new task1();

myTimer.schedule(myTask1, 100);

}

}

}

catch(Exception e)

{

System.out.println(e);

}

}

public void db()

{

try

{

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

con = DriverManager.getConnection("jdbc:odbc:projectup");

st = con.createStatement();

}catch(Exception ex)

{

System.out.println(ex);

}

}

}

//ROOT DESIGN CODING

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* rootdesign \*/

/\* \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

import java.net.\*;

import java.io.\*;

import java.util.\*;

import java.sql.\*;

import javax.swing.event.\*;

/\*\*

\* Summary description for rootdesign

\*

\*/

public class rootdesign extends JFrame

{

// Variables declaration

public static String send\_share;

private JLabel jLabel1;

private JLabel jLabel2;

private JTextField jTextField1;

private JTextField jTextField2;

private JTextArea jTextArea1;

private JScrollPane jScrollPane1;

private JButton FileTransmission;

private JButton SENDMESSAGE;

private JButton MFTREPORT;

private JButton CF;

private JPanel contentPane;

private JList jList1;

private JScrollPane jScrollPane2;

Connection con;

Statement st;

//ResultSet rs;

Vector v=new Vector();

// End of variables declaration

public rootdesign()

{

super();

try{

d obj= new d();

v=obj.nodes();

System.out.println("Vector Elements" + v);

}

catch(Exception e)

{

System.out.println("Error in nodes calling " + e);

e.printStackTrace();

}

initializeComponent();

//

// TODO: Add any constructor code after initializeComponent call

//

mains t1 = new mains();

delay t2 = new delay();

t1.start();

t2.start();

this.setVisible(true);

}

/\*\*

\* This method is called from within the constructor to initialize the form.

\* WARNING: Do NOT modify this code. The content of this method is always regenerated

\* by the Windows Form Designer. Otherwise, retrieving design might not work properly.

\* Tip: If you must revise this method, please backup this GUI file for JFrameBuilder

\* to retrieve your design properly in future, before revising this method.

\*/

private void initializeComponent()

{

jLabel1 = new JLabel();

jLabel2 = new JLabel();

jTextField1 = new JTextField();

jTextField2 = new JTextField();

jTextArea1 = new JTextArea();

jScrollPane1 = new JScrollPane();

FileTransmission = new JButton();

SENDMESSAGE = new JButton();

MFTREPORT = new JButton();

CF = new JButton();

jList1 = new JList(v);

jScrollPane2 = new JScrollPane();

contentPane = (JPanel)this.getContentPane();

//

// jLabel1

//

jLabel1.setBackground(new Color(255, 255, 255));

jLabel1.setForeground(new Color(51, 255, 51));

jLabel1.setText("SERVER NAME");

//

// jLabel2

//

jLabel2.setForeground(new Color(51, 255, 0));

jLabel2.setText("FILE LOCATION");

//

// jTextField1

//

jTextField1.setBackground(new Color(255, 255, 204));

jTextField1.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

jTextField1\_actionPerformed(e);

}

});

//

// jTextField2

//

jTextField2.setBackground(new Color(255, 255, 204));

jTextField2.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

jTextField2\_actionPerformed(e);

}

});

//

// jTextArea1

//

// jTextArea1.setText("jTextArea1");

//

// jScrollPane1

//

jScrollPane1.setViewportView(jTextArea1);

//

// FileTransmission

//

//FileTransmission.setBackground(new Color(0, 255, 204));

FileTransmission.setText("FILE TRANSMISSION");

FileTransmission.setToolTipText("IT WILL TANSMIIT THE FILE WHICH IS ENTERED IN THE TEXT AREA");

FileTransmission.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

FileTransmission\_actionPerformed(e);

}

});

//

// SENDMESSAGE

//

// SENDMESSAGE.setBackground(new Color(102, 255, 204));

SENDMESSAGE.setText("SEND MESSAGE");

SENDMESSAGE.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

SENDMESSAGE\_actionPerformed(e);

}

});

//

// MFTREPORT

//

// MFTREPORT.setBackground(new Color(51, 255, 204));

MFTREPORT.setText("MFT REPORT");

MFTREPORT.setToolTipText("SHOWS THE ROUTING TABLE MODIFICATION");

MFTREPORT.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

MFTREPORT\_actionPerformed(e);

}

});

CF.setText("CHOOSE FILE");

//MFTREPORT.setToolTipText("SHOWS THE ROUTING TABLE MODIFICATION");

CF.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

CF\_actionPerformed(e);

}

});

jList1.addListSelectionListener(new ListSelectionListener() {

public void valueChanged(ListSelectionEvent e)

{

jList1\_valueChanged(e);

}

});

jScrollPane2.setViewportView(jList1);

//

// contentPane

//

contentPane.setLayout(null);

contentPane.setBackground(new Color(0, 102, 0));

addComponent(contentPane, jLabel1, 62,81,95,29);

addComponent(contentPane, jLabel2, 60,155,92,28);

addComponent(contentPane, jTextField1, 210,79,117,33);

addComponent(contentPane, jTextField2, 212,151,115,34);

addComponent(contentPane, jScrollPane1, 399,78,225,228);

addComponent(contentPane, CF, 54,355,140,35);

addComponent(contentPane, FileTransmission, 194,355,150,35);

addComponent(contentPane, SENDMESSAGE, 344,355,131,35);

addComponent(contentPane, MFTREPORT, 475,355,130,35);

addComponent(contentPane, jScrollPane2, 680,78,120,228);

//

// rootdesign

//

this.setTitle("rootdesign - extends JFrame");

this.setLocation(new Point(0, 0));

this.setSize(new Dimension(820, 450));

try{

String ip=InetAddress.getLocalHost().getHostAddress();

System.out.println("IP ADDRESS : " + ip);

jTextField1.setText(ip);

}

catch(Exception e1)

{

System.out.println(e1);

}

}

/\*\* Add Component Without a Layout Manager (Absolute Positioning) \*/

private void addComponent(Container container,Component c,int x,int y,int width,int height)

{

c.setBounds(x,y,width,height);

container.add(c);

}

//

// TODO: Add any appropriate code in the following Event Handling Methods

//

private void jTextField1\_actionPerformed(ActionEvent e)

{

System.out.println("\njTextField1\_actionPerformed(ActionEvent e) called.");

// TODO: Add any handling code here

}

private void jTextField2\_actionPerformed(ActionEvent e)

{

System.out.println("\njTextField2\_actionPerformed(ActionEvent e) called.");

// TODO: Add any handling code here

}

private void jList1\_valueChanged(ListSelectionEvent e)

{

System.out.println("\njList1\_valueChanged(ListSelectionEvent e) called.");

if(!e.getValueIsAdjusting())

{

Object o = jList1.getSelectedValue();

System.out.println(">>" + ((o==null)? "null" : o.toString()) + " is selected.");

// TODO: Add any handling code here for the particular object being selected

}

}

private void FileTransmission\_actionPerformed(ActionEvent e)

{

System.out.println("\nFileTransmission\_actionPerformed(ActionEvent e) called.");

try{

String send1 = jTextArea1.getText();

System.out.println(mains.flag);

String s1=mains.flag;

if(s1==null)

{

delay.out2.writeUTF(send1); //for root to left (Direct path)

System.out.println("Root sending the following data to left " + send1);

}

// if(mains.flag.equals("right"))

else if(s1.equals("right"))

{

mains.out1.writeUTF(send1); //for root to right and left(Indirect path)

System.out.println("Root sending the following data to right " + send1);

}

}

catch(Exception e1)

{

System.out.println("fffffffffffff " + e1);

}

// TODO: Add any handling code here

}

private void SENDMESSAGE\_actionPerformed(ActionEvent e)

{

System.out.println("\nSENDMESSAGE\_actionPerformed(ActionEvent e) called.");

try{

String send2 = jTextArea1.getText();

System.out.println(mains.flag);

String s2=mains.flag;

if(s2==null)

{

delay.out2.writeUTF(send2); //for root to left (Direct path)

System.out.println("Root sending the following data to left " + send2);

}

// if(mains.flag.equals("right"))

else if(s2.equals("right"))

{

mains.out1.writeUTF(send2); //for root to right and left(Indirect path)

System.out.println("Root sending the following data to right " + send2);

}

}

catch(Exception e1)

{

System.out.println("Error in sending message " + e1);

}

}

private void MFTREPORT\_actionPerformed(ActionEvent e)

{

System.out.println("\n MFT REPORT\_actionPerformed (Action Event e) called.");

db();

//int i=0;

try{

reportroot r=new reportroot();

ResultSet rs4= st.executeQuery("Select \* from MFT");

String s1[]=new String[3];

String s2;

while(rs4.next())

{

s1[0]=rs4.getString(1);

s1[1]=rs4.getString(2);

s1[2]=rs4.getString(3);

//i++;

}

r.jTextField1.setText(s1[0]);

System.out.println(s1[0]);

r.jTextField2.setText(s1[1]);

System.out.println(s1[1]);

r.jTextField3.setText(s1[2]);

System.out.println(s1[2]);

con.close();

//System.out.println("after visible");

}

catch(Exception e1)

{

System.out.println("Error in MFT " + e1);

e1.printStackTrace();

}

}

private void CF\_actionPerformed(ActionEvent e)

{

System.out.println("\nfile choose\_actionPerformed(ActionEvent e) called.");

try

{

JFileChooser jf=new JFileChooser();

int m=jf.showOpenDialog(null);

if(m==JFileChooser.APPROVE\_OPTION)

{

File f1=jf.getSelectedFile();

String str=f1.getPath();

jTextField2.setText(str);

FileInputStream fis=new FileInputStream(str);

//File f=new File(str);

byte b[]=new byte[fis.available()];

fis.read(b);

String msg=new String(b);

jTextArea1.setText(msg);

fis.close();

}

}

catch(Exception ui)

{

ui.printStackTrace();

//System.err.println("Exiting program..." + finalsource.treceiver + " is not an active node");

}

}

//

// TODO: Add any method code to meet your needs in the following area

//

public void db()

{

try

{

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

con = DriverManager.getConnection("jdbc:odbc:projectup");

st = con.createStatement();

}catch(Exception ex)

{

System.out.println(ex);

}

}

//============================= Testing ================================//

//= =//

//= The following main method is just for testing this class you built.=//

//= After testing,you may simply delete it. =//

//======================================================================//

public static void main(String[] args)

{

JFrame.setDefaultLookAndFeelDecorated(true);

JDialog.setDefaultLookAndFeelDecorated(true);

try

{

UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

}

catch (Exception ex)

{

System.out.println("Failed loading L&F: ");

System.out.println(ex);

}

new rootdesign();

}

//= End of Testing =

}

LEFT DESIGN:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* LEFT \*/

/\* \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

import java.net.\*;

import java.io.\*;

import java.util.\*;

import java.sql.\*;

import javax.swing.event.\*;

import javax.swing.table.\*;

/\*\*

\* Summary description for LEFT

\*

\*/

public class LEFTDESIGN extends JFrame

{

// Variables declaration

Socket echoSocket = null;

DataInputStream dis=null;

DataOutputStream dop=null;

DataInputStream dis1=null;

DataOutputStream dop1=null;

Connection con;

Statement st;

ResultSet rs;

int left\_to\_right=0;

int left\_to\_root=0;

int right\_to\_root=0;

int indirect\_path=0;

String hostname;

private JLabel jLabel1;

private JLabel jLabel2;

private JTextField jTextField1;

private JTextField jTextField2;

private JTextArea jTextArea1;

private JScrollPane jScrollPane1;

private JButton jButton1;

private JButton jButton2;

private JButton jButton3;

private JButton jButton4;

private JPanel contentPane;

private JList jList1;

private JScrollPane jScrollPane2;

String right="right";

String righthost="";

String display;

Vector v=new Vector();

String ip;

String server;

// End of variables declaration

public LEFTDESIGN()

{

super();

try{

d obj= new d();

v=obj.nodes();

System.out.println("Vector Elements" + v);

}

catch(Exception e)

{

System.out.println("Error in nodes calling " + e);

e.printStackTrace();

}

initializeComponent();

//

// TODO: Add any constructor code after initializeComponent call

//

this.setVisible(true);

}

/\*\*

\* This method is called from within the constructor to initialize the form.

\* WARNING: Do NOT modify this code. The content of this method is always regenerated

\* by the Windows Form Designer. Otherwise, retrieving design might not work properly.

\* Tip: If you must revise this method, please backup this GUI file for JFrameBuilder

\* to retrieve your design properly in future, before revising this method.

\*/

private void initializeComponent()

{

jLabel1 = new JLabel();

jLabel2 = new JLabel();

jTextField1 = new JTextField();

jTextField2 = new JTextField();

jButton1 = new JButton();

jButton2 = new JButton();

jTextArea1 = new JTextArea();

jScrollPane1 = new JScrollPane();

jButton3 = new JButton();

jButton4 = new JButton();

jList1 = new JList(v);

jScrollPane2 = new JScrollPane();

contentPane = (JPanel)this.getContentPane();

jScrollPane1.setViewportView(jTextArea1);

//

// jLabel1

//

jLabel1.setBackground(new Color(0, 255, 51));

jLabel1.setForeground(new Color(102, 255, 255));

jLabel1.setText("HOST NAME");

//

// jLabel2

//

jLabel2.setBackground(new Color(102, 255, 255));

jLabel2.setForeground(new Color(102, 255, 255));

jLabel2.setText("SERVER NAME");

//

// jTextField1

//

jTextField1.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

jTextField1\_actionPerformed(e);

}

});

//jTextArea1.setEnabled(false);

//

// jTextField2

//

jTextField2.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

jTextField2\_actionPerformed(e);

}

});

//

// jButton1

//

//jButton1.setBackground(new Color(223, 255, 224));

jButton1.setText("REQUEST");

jButton1.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

jButton1\_actionPerformed(e);

}

});

//

// jButton2

//

//jButton2.setBackground(new Color(223, 255, 224));

jButton2.setText("MFT TABLE");

jButton2.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

jButton2\_actionPerformed(e);

}

});

//

// jButton3

//

//jButton3.setBackground(new Color(223, 255, 224));

/\*jButton3.setText("SEND MESSAGE");

jButton3.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

jButton3\_actionPerformed(e);

}

});\*/

jButton4.setText("DISPLAY");

jButton4.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e)

{

jButton4\_actionPerformed(e);

}

});

jList1.addListSelectionListener(new ListSelectionListener() {

public void valueChanged(ListSelectionEvent e)

{

jList1\_valueChanged(e);

}

});

jScrollPane2.setViewportView(jList1);

//

// contentPane

//

contentPane.setLayout(null);

contentPane.setBackground(new Color(102, 102, 0));

addComponent(contentPane, jLabel1, 68,67,93,30);

addComponent(contentPane, jLabel2, 67,127,95,28);

addComponent(contentPane, jTextField1, 206,69,101,31);

addComponent(contentPane, jTextField2, 206,131,103,30);

addComponent(contentPane, jScrollPane1, 429,78,225,228);

addComponent(contentPane, jButton1, 47,317,101,47);

addComponent(contentPane, jButton2, 148,317,99,47);

// addComponent(contentPane, jButton3, 247,317,144,47);

addComponent(contentPane, jButton4, 247,317,144,47);

addComponent(contentPane, jScrollPane2, 680,78,120,228);

//

// LEFT

//

this.setTitle("LEFT - extends JFrame");

this.setLocation(new Point(21, 8));

this.setSize(new Dimension(820, 450));

try{

db();

String ip=InetAddress.getLocalHost().getHostAddress();

System.out.println("IP ADDRESS : " + ip);

jTextField1.setText(ip);

ResultSet rs = st.executeQuery("Select \* from iptablel where locationl='"+right+"'");

// int i = 0;

while(rs.next())

{

righthost= rs.getString("hostnamel"); // take left host ip from db

System.out.println("From iptablel " + righthost);

break;

} con.close();

db();

String root="root";

ResultSet rs2 = st.executeQuery("Select \* from iptablel where locationl='"+root+"'");

// int i = 0;

while(rs2.next())

{

server= rs2.getString("hostnamel"); // take left host ip from db

System.out.println("Server name " + server);

break;

} con.close();

jTextField2.setText(server);

db();

ResultSet rs1 = st.executeQuery("Select \* from leftpath");

while(rs1.next())

{

left\_to\_right=rs1.getInt(1);

left\_to\_root=rs1.getInt(2);

right\_to\_root=rs1.getInt(3);

System.out.println("From leftpath " + left\_to\_right + " " + left\_to\_root +" " + right\_to\_root );

break;

} con.close();

indirect\_path=left\_to\_right+right\_to\_root;

}

catch(Exception e)

{

System.out.println("Error in Left " + e);

e.printStackTrace();

}

}

/\*\* Add Component Without a Layout Manager (Absolute Positioning) \*/

private void addComponent(Container container,Component c,int x,int y,int width,int height)

{

c.setBounds(x,y,width,height);

container.add(c);

}

//

// TODO: Add any appropriate code in the following Event Handling Methods

//

private void jTextField1\_actionPerformed(ActionEvent e)

{

System.out.println("\njTextField1\_actionPerformed(ActionEvent e) called.");

// TODO: Add any handling code here

}

private void jTextField2\_actionPerformed(ActionEvent e)

{

System.out.println("\njTextField2\_actionPerformed(ActionEvent e) called.");

// TODO: Add any handling code here

}

private void jList1\_valueChanged(ListSelectionEvent e)

{

System.out.println("\njList1\_valueChanged(ListSelectionEvent e) called.");

if(!e.getValueIsAdjusting())

{

Object o = jList1.getSelectedValue();

System.out.println(">>" + ((o==null)? "null" : o.toString()) + " is selected.");

// TODO: Add any handling code here for the particular object being selected

}

}

private void jButton1\_actionPerformed(ActionEvent e)

{

System.out.println("\njRequest\_actionPerformed(ActionEvent e) called.");

EventQueue.invokeLater(new Runnable() {

public void run() {

// TODO: Add any handling code here

try{

if(left\_to\_root>indirect\_path)

{

System.out.println("Shortest path is left->right->Root");

echoSocket = new Socket(righthost,1034); //socket for left to right comm 1 Socket 1034

System.out.println("Path Established From Left to Root via Right");

dis = new DataInputStream(echoSocket.getInputStream());

dop = new DataOutputStream(echoSocket.getOutputStream());

String s=InetAddress.getLocalHost().toString();

StringTokenizer st=new StringTokenizer(s,"/");

while(st.hasMoreTokens()) {

hostname = st.nextToken();

System.out.println("Address of local host is : " + hostname);

break;

}

dop.writeUTF("left");

dop.writeUTF(hostname);

// String ack=dis.readUTF();

//System.out.println("Acknoledgement : " + ack);

System.out.println("Waiting for Acknoledgement...... ");

String data\_from\_right = dis.readUTF(); // from right

display=data\_from\_right;

System.out.println("From Root " + data\_from\_right);

JOptionPane.showMessageDialog(null,"File or Message Received From Left","Group Notifier",JOptionPane.OK\_OPTION);

//System.out.println();

//jTextArea1.setText(data\_from\_right);

}

else

{

db();

String root="root";

ResultSet rs3 = st.executeQuery("Select \* from iptablel where locationl='"+root+"'");

String roothost="";

while(rs3.next())

{

//i = 1;

// String ss=rs.getString(1);

roothost= rs3.getString("hostnamel"); // take left host ip from db

System.out.println("From iptablel " + roothost);

// break;

} con.close();

System.out.println("Shortest path is left->Root");

echoSocket = new Socket(roothost,1033); // left to root comm 2 Socket 1033

System.out.println("Path established From Left to Root");

dis = new DataInputStream(echoSocket.getInputStream());

dop = new DataOutputStream(echoSocket.getOutputStream());

String s=InetAddress.getLocalHost().toString();

StringTokenizer st=new StringTokenizer(s,"/");

while(st.hasMoreTokens()) {

hostname = st.nextToken();

System.out.println("Address of local host is : " + hostname);

break;

}

dop.writeUTF("left");

dop.writeUTF(hostname);

System.out.println("Waiting for Acknoledgement...... ");

String data\_from\_root=dis.readUTF();

display=data\_from\_root;

System.out.println("From Root " + data\_from\_root);

JOptionPane.showMessageDialog(null,"File or Message Received From Root","Group Notifier",JOptionPane.OK\_OPTION);

}

}

catch(Exception e1)

{

System.out.println("Error in Left " + e1);

e1.printStackTrace();

}

}

}

);

}

private void jButton2\_actionPerformed(ActionEvent e)

{

db();

int i=0;

try{

report r=new report();

ResultSet rs4= st.executeQuery("Select \* from iptablel");

String s1[]=new String[3];

String s2;

while(rs4.next())

{

s1[i]=rs4.getString(2);

i++;

}

r.jTextField1.setText(s1[0]);

System.out.println(s1[0]);

r.jTextField2.setText(s1[1]);

System.out.println(s1[1]);

r.jTextField3.setText(s1[2]);

System.out.println(s1[2]);

con.close();

//System.out.println("after visible");

}

catch(Exception e1)

{

System.out.println("Error in MFT " + e1);

e1.printStackTrace();

}

}

private void jButton3\_actionPerformed(ActionEvent e)

{

System.out.println("\nj Button3\_actionPerformed(ActionEvent e) called.");

// TODO: Add any handling code here

}

private void jButton4\_actionPerformed(ActionEvent e)

{

System.out.println("\nj DISPALY\_actionPerformed(ActionEvent e) called.");

// TODO: Add any handling code here

jTextArea1.setEditable(true);

jTextArea1.setText(display);

System.out.println(display);

}

//

// TODO: Add any method code to meet your needs in the following area

//

public void db()

{

try

{

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

con = DriverManager.getConnection("jdbc:odbc:projectup");

st = con.createStatement();

}catch(Exception ex)

{

System.out.println(ex);

}

}

//============================= Testing ================================//

//= =//

//= The following main method is just for testing this class you built.=//

//= After testing,you may simply delete it. =//

//======================================================================//

public static void main(String[] args)

{

JFrame.setDefaultLookAndFeelDecorated(true);

JDialog.setDefaultLookAndFeelDecorated(true);

try

{

UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

}

catch (Exception ex)

{

System.out.println("Failed loading L&F: ");

System.out.println(ex);

}

new LEFTDESIGN();

}

//= End of Testing =

}

SCREEN SHOT:

ROOT DESIGN: