

# **An Adaptive Programming Model for Fault-Tolerant Distributed Computing**

## **CONTENTS**

### **1. INTRODUCTION**

1.1 About Fault Tolerance

1.2 Benefits of Fault Tolerance

### **2. ORGANIZATION PROFILE**

### **3. SYSTEM ANALYSIS**

3.1 Existing System

3.2 Limitations

3.3 Proposed System

3.4 Advantages of proposed system

### **4. PROBLEM FORMULATION**

4.1 Objectives

4.2 Hardware Specification

4.3 Software Specification

4.4 Software Descriptions

### **5. SYSTEM DESIGN**

5.1 Design Overview

5.2 Context Analysis Diagram

5.3 Data Flow Diagram

5.4 Architecture Design

6.

### **SYSTEM TESTING**

6.1 Unit Testing

6.2 Integration Testing

## 6.3 Acceptance Testing

### 7. SYSTEM IMPLEMENTATION

### 8. CONCLUSION

### 9. FUTURE ENCHANCEMENTS

### 10. BIBLIOGRAPHY

### 11. APPENDICES

### APPENDIX A: SAMPLE SCREENS

## 1. INTRODUCTION

### 1.1 About Fault Tolerance:

DISTRIBUTED systems are composed of processes, located on one or more sites that communicate with one another to offer services to upper-layer applications. A major difficulty a system designer has to cope with in these systems lies in the capture of consistent global states from which safe decisions can be taken in order to guarantee a safe progress of the upper-layer applications. To study and investigate what can be done (and how it has to be done) in these systems when they are prone to process failures, two distributed computing models have received significant attention, namely, the synchronous model and the asynchronous model.

The synchronous distributed computing model provides processes with bounds on processing time and message transfer delay. These bounds, explicitly known by the processes, can be used to safely detect process crashes and, consequently, allow the noncrashed processes to progress with safe views of the system state (such views can be obtained with some “time-lag”). In contrast, the asynchronous model is characterized by the absence of time bounds (that is why this model is sometimes called the time-free model). In these systems, a system designer can only assume an upper bound on the number of processes that can crash (usually denoted as  $f$ ) and, consequently, design protocols relying on the assumption that at least  $(n - f)$  processes are alive ( $n$  being the total number of processes). The protocol has no means to know whether a given process is alive or not. Moreover, if more than  $f$  processes crash,

there is no guarantee on the protocol behavior (usually, the protocol loses its liveness property).

Synchronous systems are attractive because they allow system designers to solve many problems. The price that has to be paid is the a priori knowledge on time bounds. If they are violated, the upper-layer protocols may be unable to still guarantee their safety property. As they do not rely on explicit time bounds, asynchronous systems do not have this drawback. Unfortunately, they have another one, namely, some basic problems are impossible to solve in asynchronous systems. The most famous is the consensus problem that has no deterministic solution when even a single process can crash [12].

The consensus problem can be stated as follows: Each process proposes a value, and has to decide a value, unless it crashes (termination), such that there is a single decided value (uniform agreement), and that value is a proposed value (validity). This problem, whose statement is particularly simple, is fundamental in fault-tolerant distributed computing as it abstracts several basic agreement problems. While consensus is considered as a “theoretical” problem, system designers are usually interested in the more practical Atomic Broadcast problem. That problem is both a communication problem and an agreement problem. Its communication part specifies that the processes can broadcast and deliver messages in such a way that the processes that do not crash deliver at least the messages they send. Its agreement part specifies that there is a single delivery order (so, the correct processes deliver the same sequence of messages, and a faulty process delivers a prefix of this sequence of messages). It has been shown that consensus and atomic broadcast are equivalent problems in asynchronous systems prone to process crashes [6]. Consequently, in asynchronous distributed systems prone to process crashes, the impossibility of solving consensus extends to atomic broadcast. This impossibility has motivated researchers to find distributed computing models, weaker than the synchronous model but stronger than the asynchronous model, in which consensus can be solved.

## **1.2 Benefits of Fault Tolerance:**

The various benefits of using Fault tolerance are listed below,

Our model provides upper-layer applications with process state information according to the current system synchrony (or QoS).

Failure is reduced compare to other existing systems.

It will be applicable to both Synchrony and Asynchrony.

Retransmission is less compare to the existing systems.

It increase the throughputs in network

Not cost oriented.

Reduced administrative costs.

Lower error rates.

Increased Productivity.

## **2. ORGANIZATION PROFILE**

### **COMPANY PROFILE.**

At Blue Chip Technologies, We go beyond providing software solutions. We work with our client's technologies and business changes that shape their competitive advantages.

Founded in 2000, Blue Chip Technologies (P) Ltd. is a software and service provider that helps organizations deploy, manage, and support their business-critical software more effectively. Utilizing a combination of proprietary software, services and specialized expertise, Blue Chip Technologies (P) Ltd. helps mid-to-large enterprises, software companies and IT service providers improve consistency, speed, and transparency with service delivery at lower costs. Blue Chip Technologies (P) Ltd. helps companies avoid many of the delays, costs and risks associated with the distribution and support of software on desktops, servers and remote devices. Our automated solutions include rapid, touch-free deployments, ongoing software upgrades, fixes and security patches, technology asset inventory and tracking, software license optimization, application self-healing and policy management. At Blue Chip Technologies, we go beyond providing software solutions. We work with our clients' technologies and business processes that shape there competitive advantages.

### **About The People**

As a team we have the prowess to have a clear vision and realize it too. As a statistical evaluation, the team has more than 40,000 hours of expertise in providing real-time solutions in the fields of Embedded Systems, Control

systems, Micro-Controllers, c Based Interfacing, Programmable Logic Controller, VLSI Design And Implementation, Networking With C, ++, java, client Server Technologies in Java,(J2EE\J2ME\J2SE\EJB),VB & VC++, Oracle and operating system concepts with LINUX.

### **Our Vision**

“Dreaming a vision is possible and realizing it is our goal”.

### **Our Mission**

We have achieved this by creating and perfecting processes that are in par with the global standards and we deliver high quality, high value services, reliable and cost effective IT products to clients around the world.

### **Clientele.**

- *Aray InfoTech*
- *Inquirre consultancy (U.S.A)*
- *K square consultancy pvt Ltd (U.S.A)*
- *Opal solutions*
- *Texlab Solutions*
- *Vertex Business Machines*
- *JM InfoTech*

### **Existing System:**

The synchronous distributed computing model provides processes with bounds on processing time and message transfer delay. These bounds, explicitly known by the processes, can be used to safely detect process crashes and, consequently, allow the noncrashed processes to progress with safe views of the system state (such views can be obtained with some “time-lag”). In contrast, the asynchronous model is characterized by the absence of time bounds (that is why this model is sometimes called the time-free model). In these systems, a system designer can only assume an upper bound on the number of processes that can crash (usually denoted as  $f$ ) and, consequently, design protocols relying on the assumption that at least  $(n - f)$  processes are alive ( $n$  being the total number of processes). The protocol has no means to know whether a given process is alive or not. Moreover, if more than  $f$  processes crash, there is no guarantee on the protocol behavior (usually, the protocol loses its liveness property).

## **Proposed System:**

Our programming model provides the upper-layer applications with sufficient process state information (the sets) that can be used in order to adapt to the available system synchrony or QoS (in terms of timely and untimely channels), providing more efficient solutions to fault tolerant problems when possible.

Implementing our hybrid programming model requires some basic facilities such as the provision and monitoring of QoS communications with both bounded and unbounded delivery times, and also a mechanism to adapt the system when timely channels can no longer be guaranteed, due to failures. Our programming model builds on facilities typically encountered in QoS architectures, such as Omega, QoS-A, Quartz, and Differentiated Services. In particular, we assume that the underlying system is capable of providing timely communication channels (alike services such as QoS hard, deterministic, and Express Forward). Similarly, we assume the existence of best-effort channels where messages are transmitted without guaranteed bounded time delays. We call these channels untimely. QoS monitoring and fail-awareness have been implemented by the QoS Provider (QoSP), failure and state detectors mechanisms, briefly presented below. It was a design decision to build our model on top of a QoS-based system. However, we could also have implemented our programming model based on facilities encountered in existing hybrid architectures: For instance, timely channels could be implemented using RTD channels by setting the probabilities  $P_d$  (deadline probability) and  $P_r$  (reliability probability) close to one, and untimely channels could be implemented with a basic channel without any guarantees [19]. The timing failure detection service of TCB [4] could then complement the required functionality.

## **Modules:**

- Identify the status of Node

- Message Transmission

- Change status

- Update status

## **Identify the Status Node:**

In this Module we identify the Node is weather live or not. In this process we easily identify the status of the node and also easily identify the path failure.

Message Transmission:

In the module we just transfer the message to the destination or intermediate nodes.

The intermediate node just forwards the message to destination.

The receiver receives the message and sends the Ack.

Change Status:

In this Module we identify the changed status of node. The Status is

- Live
- Uncertain
- Down

Update Status:

In this module we update the status of the node. Then only we can identify whether the node is live or not.

## 4. PROBLEM FORMULATION

### The Consensus Problem:

In the consensus problem, every correct process  $p_i$  proposes a value  $v_i$  and all correct processes have to decide on the same value  $v$ , that has to be one of the proposed values. More precisely, the Consensus problem is defined by two safety properties (Validity and Uniform Agreement) and a Termination Property [6], [12]:

Validity: If a process decides  $v$ , then  $v$  was proposed by some process.

Uniform Agreement: No two processes decide differently.

Termination: Every correct process eventually decides on some value.

### 4.1 Objectives

- This paper proposed and fully developed an adaptive programming model for fault-tolerant distributed computing. Our model provides upper-layer applications with process state information according to the current system synchrony (or QoS).

### 4.2 HARDWARE SPECIFICATION

<b>Processor</b>	: Any Processor above 500 Mhz.
<b>Ram</b>	: 128Mb.
<b>Hard Disk</b>	: 10 Gb.
<b>Compact Disk</b>	: 650 Mb.

**Input device** : Standard Keyboard and Mouse.  
**Output device** : VGA and High Resolution Monitor.

### 4.3 SOFTWARE SPECIFICATION

**Operating System** : Windows 2000 server Family.  
**Techniques** : JDK 1.5  
**Data Bases** : Microsoft Sql Server  
**Front End** : Java Swing  
**Back End** : Sql Server

### 4.4 Software Environment

#### Java Technology

Java technology is both a programming language and a platform.

#### The Java Programming Language

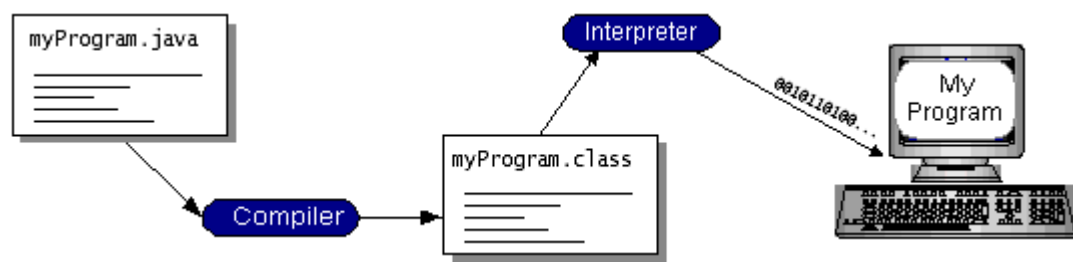
The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

- Simple
- Architecture neutral
- Object oriented
- Portable
- Distributed
- High performance
- Interpreted
- Multithreaded
- Robust
- Dynamic
- Secure

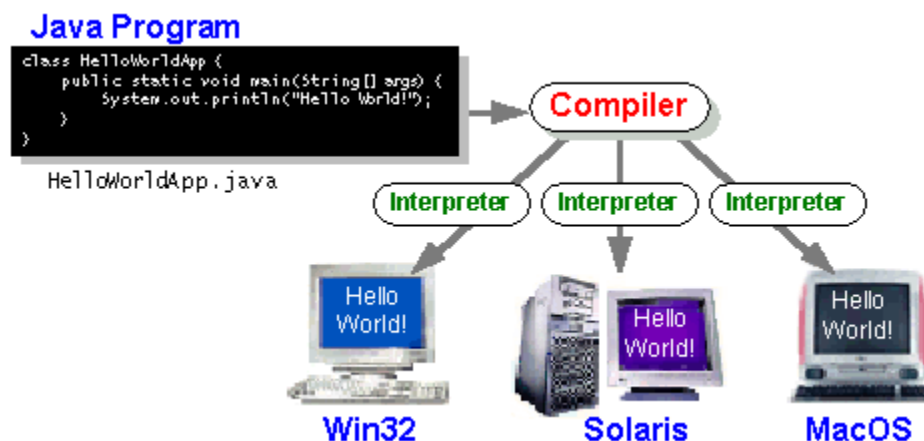
With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called *Java byte codes* —the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer.



Compilation happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works.



You can think of Java bytecodes as the machine code instructions for the *Java Virtual Machine* (Java VM). Every Java interpreter, whether it's a development tool or a Web browser that can run applets, is an implementation of the Java VM. Java bytecodes help make “write once, run anywhere” possible. You can compile your program into bytecodes on any platform that has a Java compiler. The bytecodes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac.



## The Java Platform

A *platform* is the hardware or software environment in which a program runs. We've already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

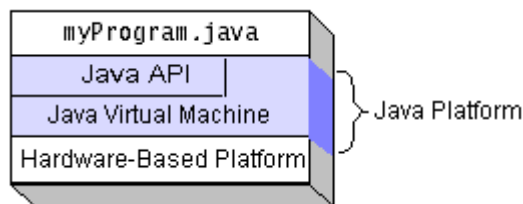
The Java platform has two components:

- The *Java Virtual Machine* (Java VM)
- The *Java Application Programming Interface* (Java API)

You've already been introduced to the Java VM. It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as *packages*. The next section, *What Can Java Technology Do?*, highlights what functionality some of the packages in the Java API provide.

The following figure depicts a program that's running on the Java platform. As the figure shows, the Java API and the virtual machine insulate the program from the hardware.



Native code is code that after you compile it, the compiled code runs on a specific hardware platform. As a platform-independent environment, the Java platform can be a bit slower than native code. However, smart compilers, well-tuned interpreters, and just-in-time bytecode compilers can bring performance close to that of native code without threatening portability.

### ***What Can Java Technology Do?***

The most common types of programs written in the Java programming language are *applets* and *applications*. If you've surfed the Web, you're probably already familiar with applets. An applet is a program that adheres to certain conventions that allow it to run within a Java-enabled browser.

However, the Java programming language is not just for writing cute, entertaining applets for the Web. The general-purpose, high-level Java programming language is also a powerful software platform. Using the generous API, you can write many types of programs.

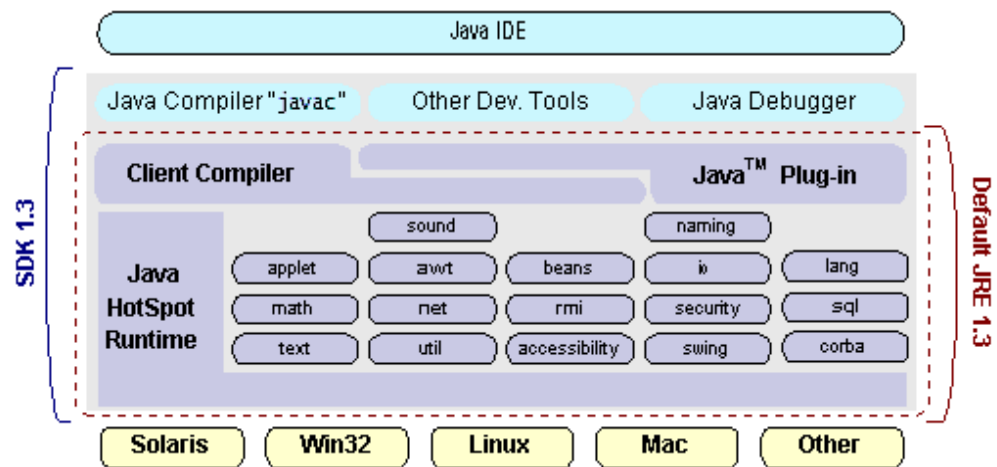
An application is a standalone program that runs directly on the Java platform. A special kind of application known as a *server* serves and supports clients on

a network. Examples of servers are Web servers, proxy servers, mail servers, and print servers. Another specialized program is a *servlet*. A servlet can almost be thought of as an applet that runs on the server side. Java Servlets are a popular choice for building interactive web applications, replacing the use of CGI scripts. Servlets are similar to applets in that they are runtime extensions of applications. Instead of working in browsers, though, servlets run within Java Web servers, configuring or tailoring the server.

How does the API support all these kinds of programs? It does so with packages of software components that provide a wide range of functionality. Every full implementation of the Java platform gives you the following features:

- **The essentials:** Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
- **Applets:** The set of conventions used by applets.
- **Networking:** URLs, TCP (Transmission Control Protocol), UDP (User Datagram Protocol) sockets, and IP (Internet Protocol) addresses.
- **Internationalization:** Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.
- **Security:** Both low level and high level, including electronic signatures, public and private key management, access control, and certificates.
- **Software components:** Known as JavaBeans<sup>TM</sup>, can plug into existing component architectures.
- **Object serialization:** Allows lightweight persistence and communication via Remote Method Invocation (RMI).
- **Java Database Connectivity (JDBC<sup>TM</sup>):** Provides uniform access to a wide range of relational databases.

The Java platform also has APIs for 2D and 3D graphics, accessibility, servers, collaboration, telephony, speech, animation, and more. The following figure depicts what is included in the Java 2 SDK.



## *How Will Java Technology Change My Life?*

We can't promise you fame, fortune, or even a job if you learn the Java programming language. Still, it is likely to make your programs better and requires less effort than other languages. We believe that Java technology will help you do the following:

- **Get started quickly:** Although the Java programming language is a powerful object-oriented language, it's easy to learn, especially for programmers already familiar with C or C++.
- **Write less code:** Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in the Java programming language can be four times smaller than the same program in C++.
- **Write better code:** The Java programming language encourages good coding practices, and its garbage collection helps you avoid memory leaks. Its object orientation, its JavaBeans component architecture, and its wide-ranging, easily extendible API let you reuse other people's tested code and introduce fewer bugs.
- **Develop programs more quickly:** Your development time may be as much as twice as fast versus writing the same program in C++. Why? You write fewer lines of code and it is a simpler programming language than C++.
- **Avoid platform dependencies with 100% Pure Java:** You can keep your program portable by avoiding the use of libraries written in other languages. The 100% Pure Java™ Product Certification Program has a

repository of historical process manuals, white papers, brochures, and similar materials online.

- **Write once, run anywhere:** Because 100% Pure Java programs are compiled into machine-independent bytecodes, they run consistently on any Java platform.
- **Distribute software more easily:** You can upgrade applets easily from a central server. Applets take advantage of the feature of allowing new classes to be loaded “on the fly,” without recompiling the entire program.

## ODBC

Microsoft Open Database Connectivity (ODBC) is a standard programming interface for application developers and database systems providers. Before ODBC became a *de facto* standard for Windows programs to interface with database systems, programmers had to use proprietary languages for each database they wanted to connect to. Now, ODBC has made the choice of the database system almost irrelevant from a coding perspective, which is as it should be. Application developers have much more important things to worry about than the syntax that is needed to port their program from one database to another when business needs suddenly change.

Through the ODBC Administrator in Control Panel, you can specify the particular database that is associated with a data source that an ODBC application program is written to use. Think of an ODBC data source as a door with a name on it. Each door will lead you to a particular database. For example, the data source named Sales Figures might be a SQL Server database, whereas the Accounts Payable data source could refer to an Access database. The physical database referred to by a data source can reside anywhere on the LAN.

The ODBC system files are not installed on your system by Windows 95. Rather, they are installed when you setup a separate database application, such as SQL Server Client or Visual Basic 4.0. When the ODBC icon is installed in Control Panel, it uses a file called ODBCINST.DLL. It is also possible to administer your ODBC data sources through a stand-alone program called ODBCADM.EXE. There is a 16-bit and a 32-bit version of this program, and each maintains a separate list of ODBC data sources.

From a programming perspective, the beauty of ODBC is that the application can be written to use the same set of function calls to interface with any data source, regardless of the database vendor. The source code of the application doesn't change whether it talks to Oracle or SQL Server. We only mention these two as an example. There are ODBC drivers available for several dozen popular database systems. Even Excel spreadsheets and plain text files can be turned into data sources. The operating system uses the Registry information written by ODBC Administrator to determine which low-level ODBC drivers are needed to talk to the data source (such as the interface to Oracle or SQL Server). The loading of the ODBC drivers is transparent to the ODBC application program. In a client/server environment, the ODBC API even handles many of the network issues for the application programmer.

The advantages of this scheme are so numerous that you are probably thinking there must be some catch. The only disadvantage of ODBC is that it isn't as efficient as talking directly to the native database interface. ODBC has had many detractors make the charge that it is too slow. Microsoft has always claimed that the critical factor in performance is the quality of the driver software that is used. In our humble opinion, this is true. The availability of good ODBC drivers has improved a great deal recently. And anyway, the criticism about performance is somewhat analogous to those who said that compilers would never match the speed of pure assembly language. Maybe not, but the compiler (or ODBC) gives you the opportunity to write cleaner programs, which means you finish sooner. Meanwhile, computers get faster every year.

## **JDBC**

In an effort to set an independent database standard API for Java, Sun Microsystems developed Java Database Connectivity, or JDBC. JDBC offers a generic SQL database access mechanism that provides a consistent interface to a variety of RDBMSs. This consistent interface is achieved through the use of "plug-in" database connectivity modules, or *drivers*. If a database vendor wishes to have JDBC support, he or she must provide the driver for each platform that the database and Java run on.

To gain a wider acceptance of JDBC, Sun based JDBC's framework on ODBC. As you discovered earlier in this chapter, ODBC has widespread support on a variety of platforms. Basing JDBC on ODBC will allow vendors to bring JDBC

drivers to market much faster than developing a completely new connectivity solution.

JDBC was announced in March of 1996. It was released for a 90 day public review that ended June 8, 1996. Because of user input, the final JDBC v1.0 specification was released soon after.

The remainder of this section will cover enough information about JDBC for you to know what it is about and how to use it effectively. This is by no means a complete overview of JDBC. That would fill an entire book.

## **JDBC Goals**

Few software packages are designed without goals in mind. JDBC is one that, because of its many goals, drove the development of the API. These goals, in conjunction with early reviewer feedback, have finalized the JDBC class library into a solid framework for building database applications in Java.

The goals that were set for JDBC are important. They will give you some insight as to why certain classes and functionalities behave the way they do. The eight design goals for JDBC are as follows:

### **1. *SQL Level API***

The designers felt that their main goal was to define a SQL interface for Java. Although not the lowest database interface level possible, it is at a low enough level for higher-level tools and APIs to be created. Conversely, it is at a high enough level for application programmers to use it confidently. Attaining this goal allows for future tool vendors to “generate” JDBC code and to hide many of JDBC’s complexities from the end user.

### **2. *SQL Conformance***

SQL syntax varies as you move from database vendor to database vendor. In an effort to support a wide variety of vendors, JDBC will allow any query statement to be passed through it to the underlying database driver. This allows the connectivity module to handle non-standard functionality in a manner that is suitable for its users.

### ***3. JDBC must be implemental on top of common database interfaces***

The JDBC SQL API must “sit” on top of other common SQL level APIs. This goal allows JDBC to use existing ODBC level drivers by the use of a software interface. This interface would translate JDBC calls to ODBC and vice versa.

### ***4. Provide a Java interface that is consistent with the rest of the Java system***

Because of Java’s acceptance in the user community thus far, the designers feel that they should not stray from the current design of the core Java system.

### ***5. Keep it simple***

This goal probably appears in all software design goal listings. JDBC is no exception. Sun felt that the design of JDBC should be very simple, allowing for only one method of completing a task per mechanism. Allowing duplicate functionality only serves to confuse the users of the API.

### ***6. Use strong, static typing wherever possible***

Strong typing allows for more error checking to be done at compile time; also, less errors appear at runtime.

### ***7. Keep the common cases simple***

Because more often than not, the usual SQL calls used by the programmer are simple `SELECT`’s, `INSERT`’s, `DELETE`’s and `UPDATE`’s, these queries should be simple to perform with JDBC. However, more complex SQL statements should also be possible.

Finally we decided to proceed the implementation using [Java Networking](#).

And for dynamically updating the cache table we go for [MS Access](#) database .

Java ha two things: a programming language and a platform. Java is a high-level programming language that is all of the following

Simple

Architecture-neutral

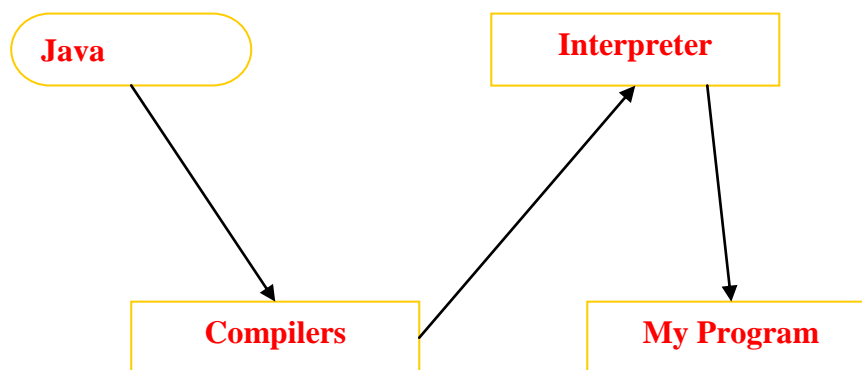
Object-oriented



Portable  
Distributed  
High-performance  
Interpreted  
multithreaded  
Robust  
Dynamic  
Secure

Java is also unusual in that each Java program is both compiled and interpreted. With a compiler you translate a Java program into an intermediate language called Java byte codes the platform-independent code instruction is passed and run on the computer.

Compilation happens just once; interpretation occurs each time the program is executed. The figure illustrates how this works.



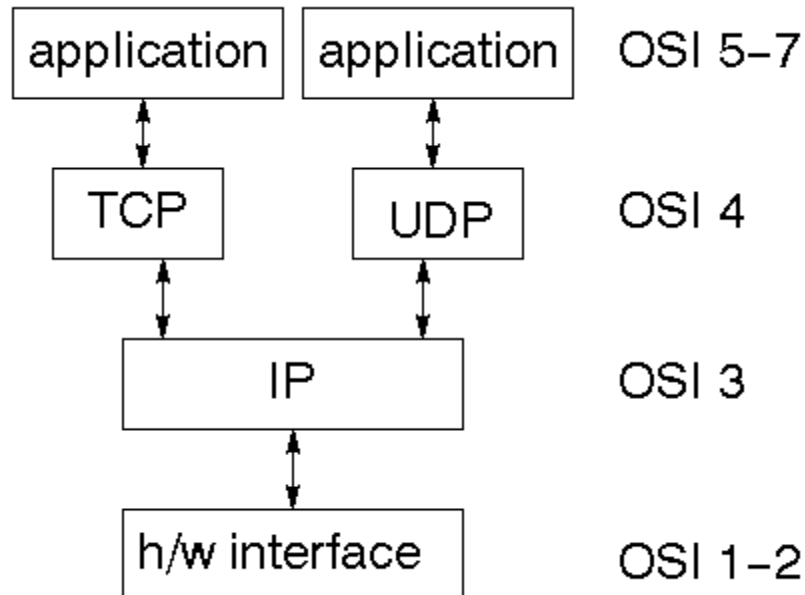
You can think of Java byte codes as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it's a Java development tool or a Web browser that can run Java applets, is an implementation of the Java VM. The Java VM can also be implemented in hardware.

Java byte codes help make "write once, run anywhere" possible. You can compile your Java program into byte codes on my platform that has a Java compiler. The byte codes can then be run any implementation of the Java VM. For example, the same Java program can run Windows NT, Solaris, and Macintosh.

## Networking

### 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.

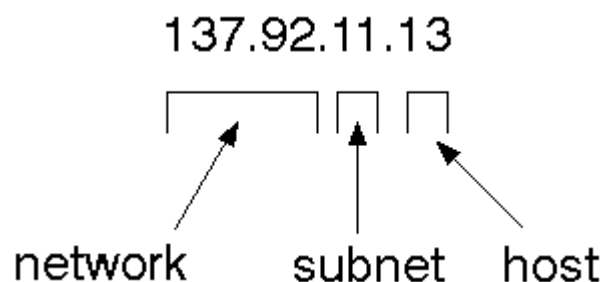
### **Subnet address**

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 `Read File` and `Write File` functions.

```
#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.
```

## JFree Chart

JFreeChart is a free 100% Java chart library that makes it easy for developers to display professional quality charts in their applications. JFreeChart's extensive feature set includes:

- a consistent and well-documented API, supporting a wide range of chart types;
- a flexible design that is easy to extend, and targets both server-side and client-side applications;
- support for many output types, including Swing components, image files (including PNG and JPEG), and vector graphics file formats (including PDF, EPS and SVG);

- JFreeChart is "open source" or, more specifically, [free software](#). It is distributed under the terms of the [GNU Lesser General Public Licence](#) (LGPL), which permits use in proprietary applications.

### ***1. Map Visualizations***

Charts showing values that relate to geographical areas. Some examples include: (a) population density in each state of the United States, (b) income per capita for each country in Europe, (c) life expectancy in each country of the world. The tasks in this project include:

- sourcing freely redistributable vector outlines for the countries of the world, states/provinces in particular countries (USA in particular, but also other areas);
- creating an appropriate dataset interface (plus default implementation), a rendered, and integrating this with the existing XYPlot class in JFreeChart;
- Testing, documenting, testing some more, documenting some more.

### ***2. Time Series Chart Interactivity***

Implement a new (to JFreeChart) feature for interactive time series charts --- to display a separate control that shows a small version of ALL the time series data, with a sliding "view" rectangle that allows you to select the subset of the time series data to display in the main chart.

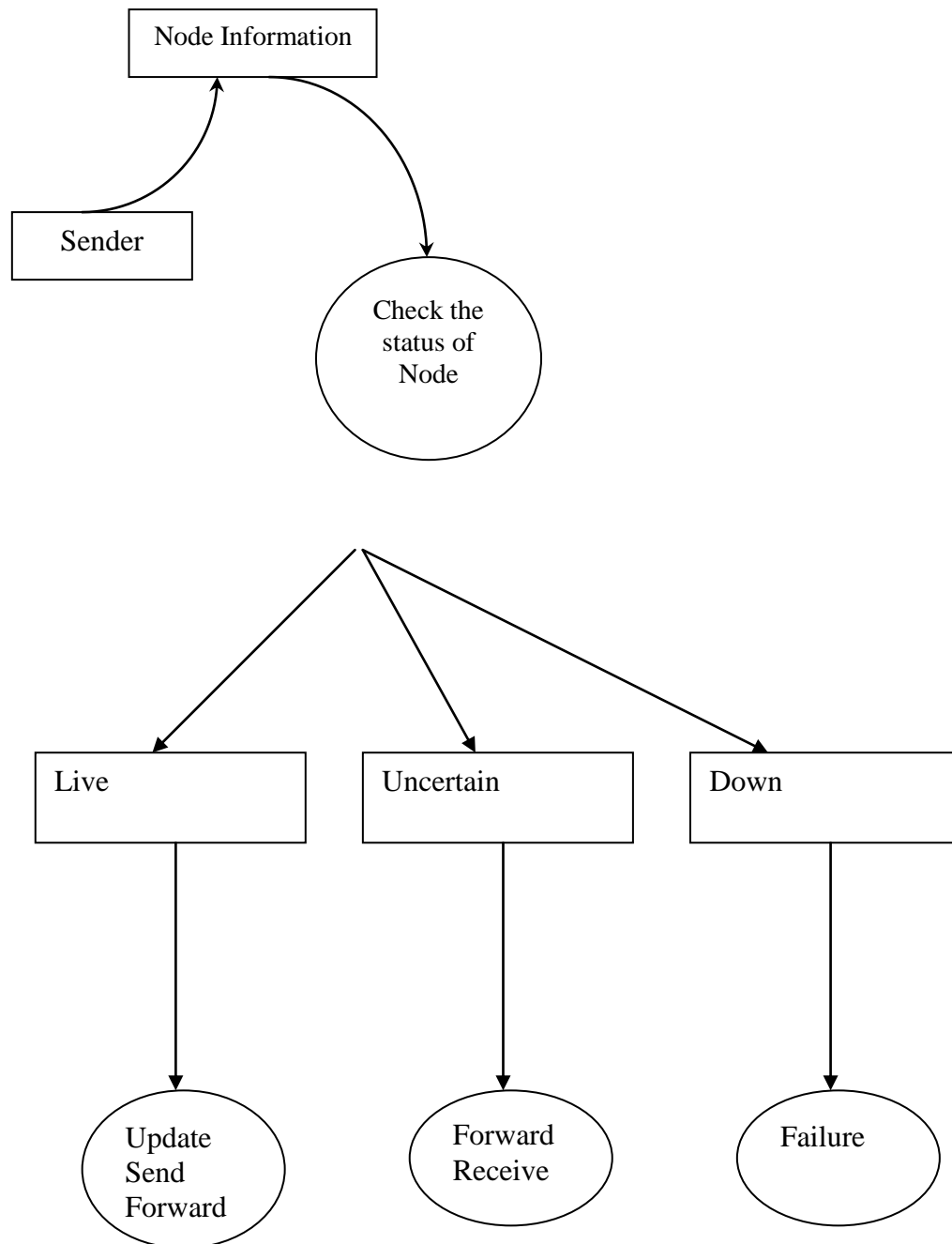
### ***3. Dashboards***

There is currently a lot of interest in dashboard displays. Create a flexible dashboard mechanism that supports a subset of JFreeChart chart types (dials, pies, thermometers, bars, and lines/time series) that can be delivered easily via both Java Web Start and an applet.

#### 4. Property Editors

The property editor mechanism in JFreeChart only handles a small subset of the properties that can be set for charts. Extend (or reimplement) this mechanism to provide greater end-user control over the appearance of the charts.

#### Data Flow Diagram



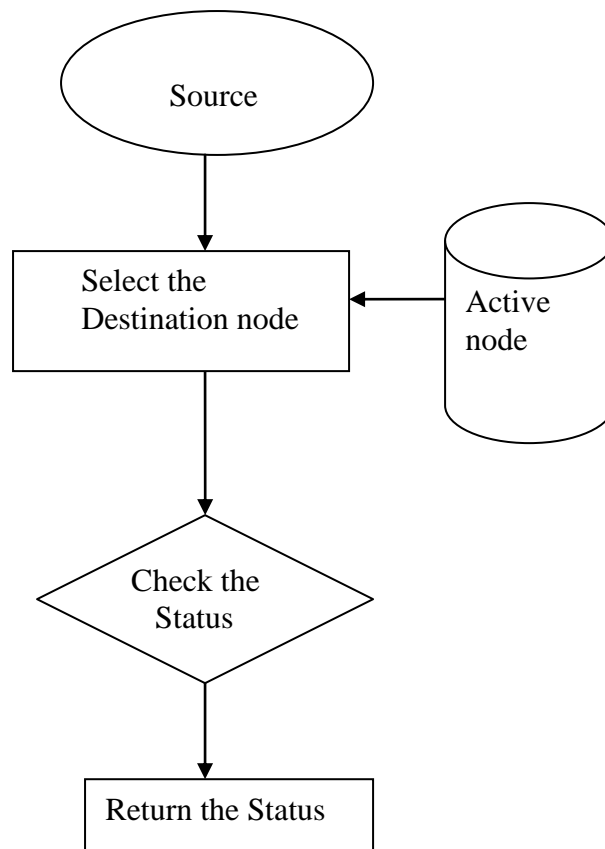
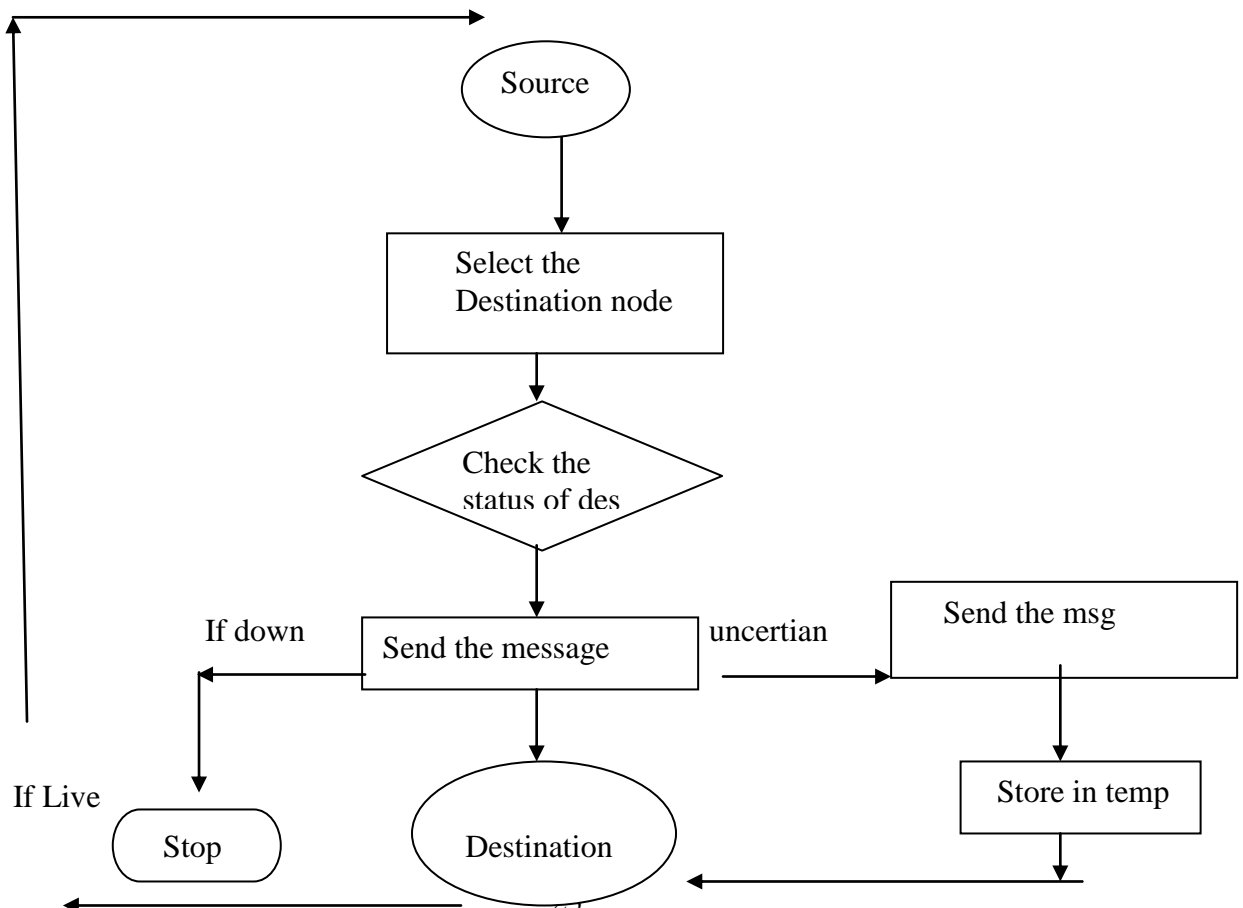


fig :Status of the node

Fig Message Transmission



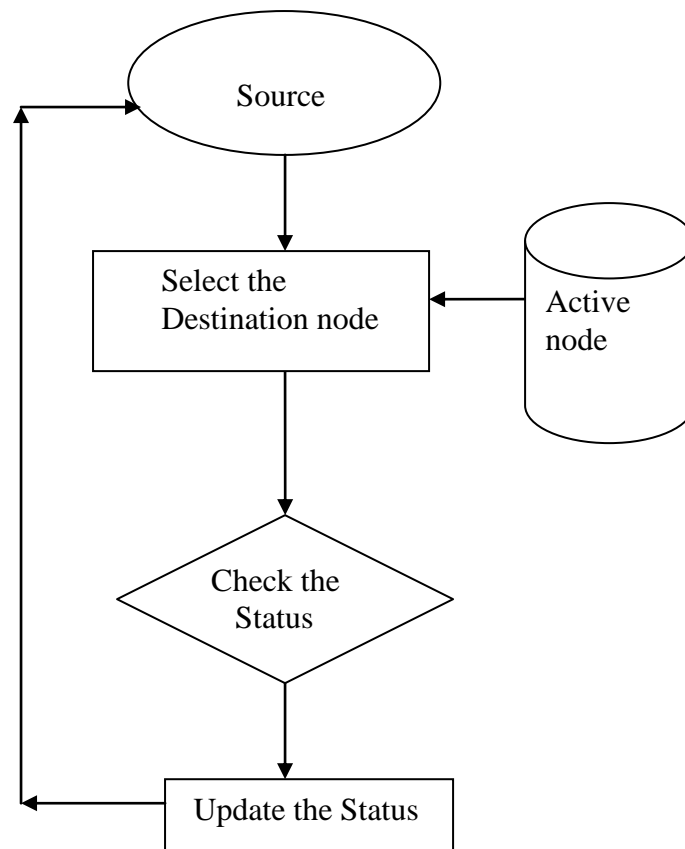
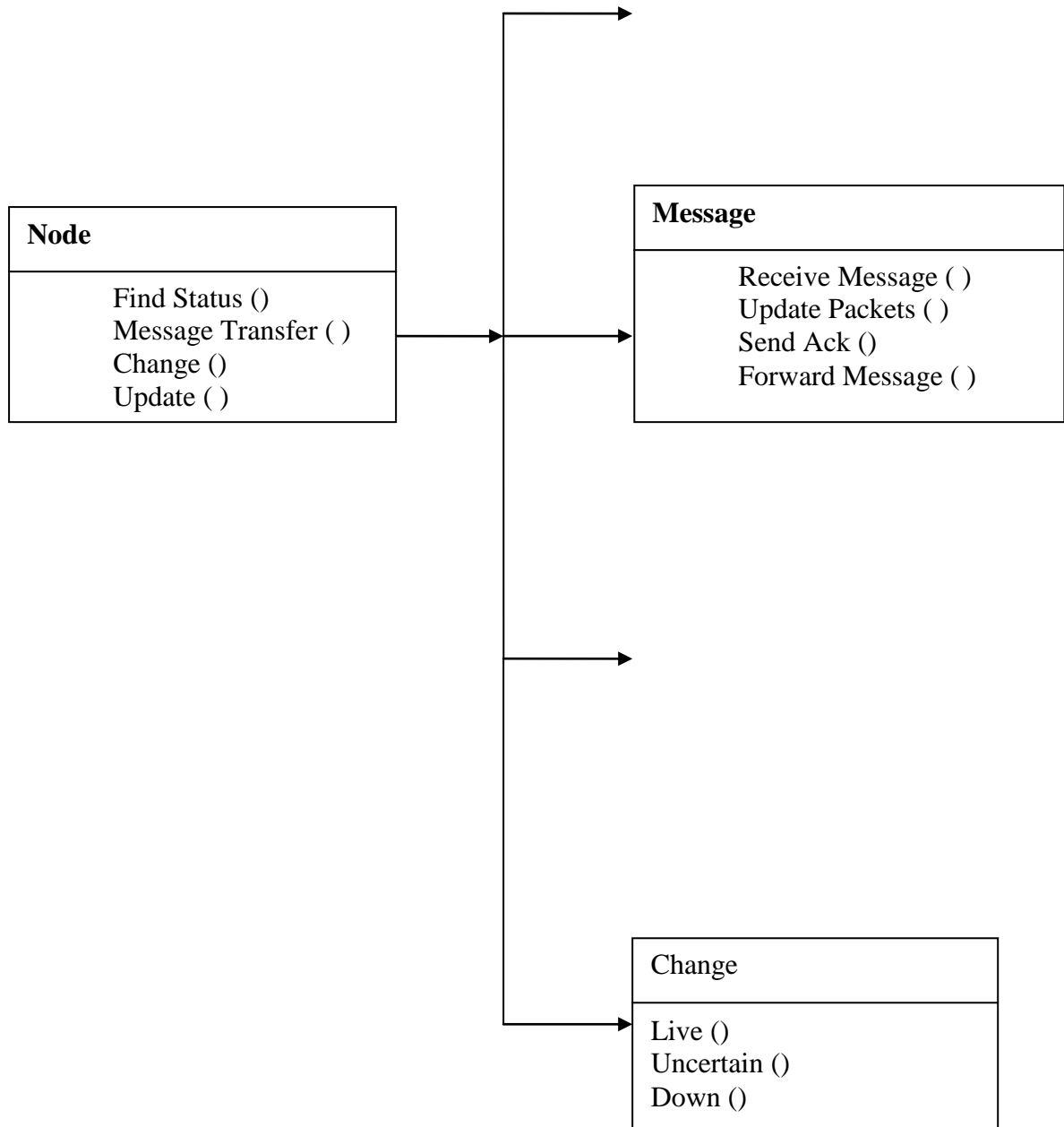


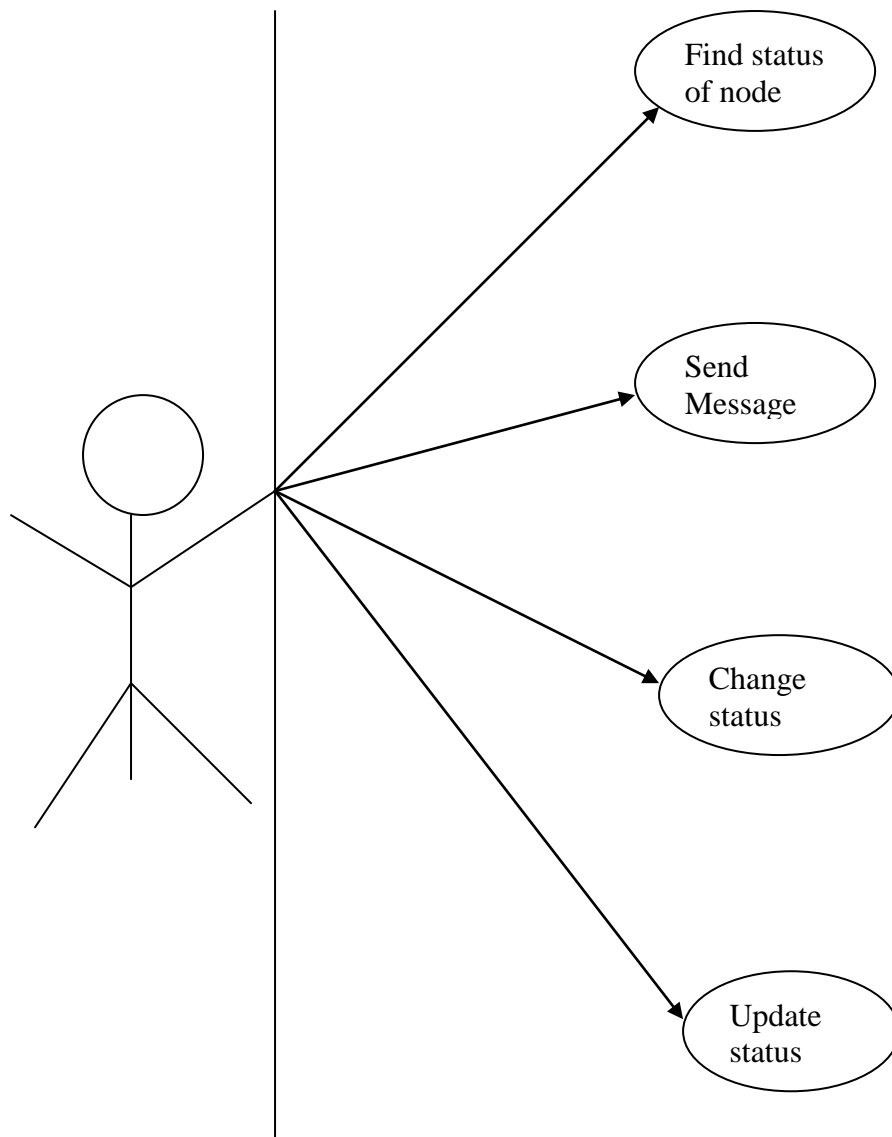
Fig Change the Status

Find
Find Status of node () Receive ack ()

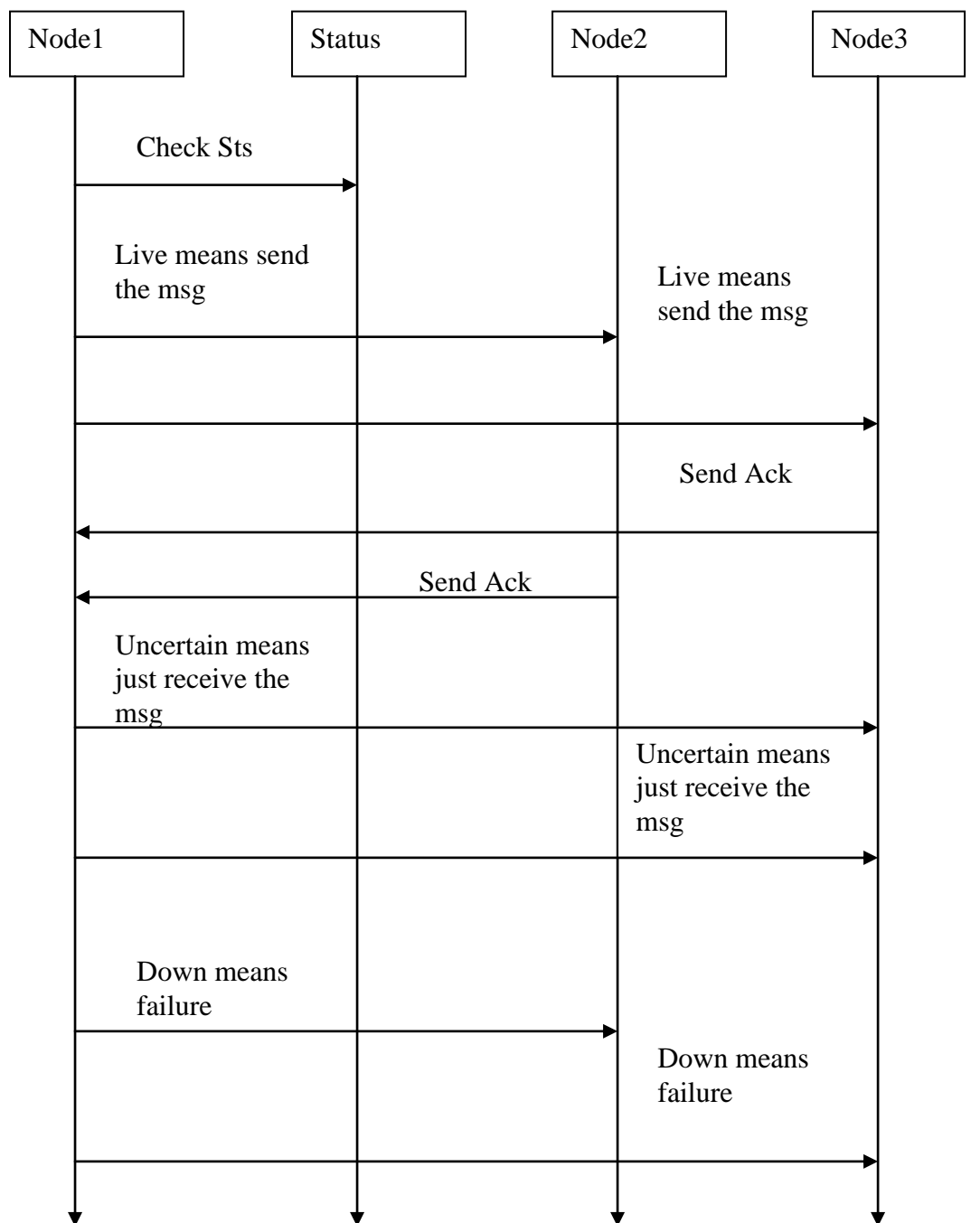
Update
Receive Status () Update Table () Forward Status ()







## Use case diagram



## **sequence diagram**

# **SYSTEM DESIGN**

## **5.1 Design Overview**

Design involves identification of classes, their relationships as well as their collaboration. In objectory ,classes were divided into Entity classes ,interface classes and the control classes. The Computer Aided Software Engineering tools that are available commercially do not provide any assistance in this transition. Even research CASE tools take advantage of meta modeling are helpful only after the construction of class diagram is completed. In the Fusion method ,it used some object-oriented approaches like Object Modeling Technique(OMT),Class\_Responsibility\_Collaborator(CRC) and Objectory,used the term Agents to represent some of the hardware and software systems .In Fusion method, there was no requirement phase ,where in a user will supply the initial requirement document. Any software project is worked out by both analyst and designer. The analyst creates the Use case diagram. The designer creates the Class diagram. But the designer can do this only after the analyst has created the Use case diagram. Once the design is over it is need to decide which software is suitable for the application

## **6. SYSTEM TESTING**

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.

### **TYPES OF TESTS**

#### **Unit testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program input produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software

units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

## **Integration testing**

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

## **Functional test**

Functional tests provide a systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation , and user manuals.

Functional testing is centered on the following items:

- Valid Input : identified classes of valid input must be accepted.
- Invalid Input : identified classes of invalid input must be rejected.
- Functions : identified functions must be exercised.
- Output : identified classes of application outputs must be exercised.
- Systems/Procedures : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

## **System Test**

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

## **White Box Testing**

White Box Testing is a testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is used to test areas that cannot be reached from a black box level .

## **Black Box Testing**

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested . Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

## **6.1 Unit Testing:**

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

### ***Test strategy and approach***

Field testing will be performed manually and functional tests will be written in detail.

### **Test objectives**

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

### **Features to be tested**

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

## **6.2 Integration Testing**

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

**Test Results:**

All the test cases mentioned above passed successfully. No defects encountered.

**6.3 Acceptance Testing**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

**Test Results:**

All the test cases mentioned above passed successfully. No defects encountered.

**7. IMPLEMENTATION**

Implementation is the stage of the project when the theoretical design is turned out into a working system. Thus it can be considered to be the most critical stage in achieving a successful new system and in giving the user, confidence that the new system will work and be effective.

The implementation stage involves careful planning, investigation of the existing system and its constraints on implementation, designing of methods to achieve changeover and evaluation of changeover methods.

Implementation is the process of converting a new system design into operation. It is the phase that focuses on user training, site preparation and file conversion for installing a candidate system. The important factor that should be considered here is that the conversion should not disrupt the functioning of the organization.

The implementation can be preceded through Socket in java but it will be considered as one to all communication .For proactive broadcasting we need dynamic linking. So java will be more suitable for platform independence and networking concepts. For maintaining route information we go for SQL-server as database back end.

**8. CONCLUSION**

This paper proposed and fully developed an adaptive programming model for fault-tolerant distributed computing. Our model provides upper-layer applications with

process state information according to the current system synchrony (or QoS). The underlying system model is hybrid, comprised of a synchronous part and an asynchronous part. However, such a composition can vary over time in such a way that the system may become totally synchronous or totally asynchronous.

The programming model is given by three sets (processes perceive each other's states by accessing the contents of their local nonintersecting sets—uncertain, live, and down) and the rules R0-R6 that regulate modifications on the sets. Moreover, we showed how those rules and the sets can be implemented in real systems.

To illustrate the adaptive ness of our model, we developed a consensus algorithm that makes progress despite distinct views of the corresponding local sets, and can tolerate more faults, the more processes are in the live set. The presented consensus algorithm adapts to the current QoS available (via the sets) and uses the majority assumption only when needed.

## **9. Future Enhancement**

## **10. BIBLIOGRAPHY**

Good Teachers are worth more than thousand books, we have them in Our Department

### **References Made From :**

1. Professional Java Network Programming
2. Java Complete Reference

[1] C. Aurrecochea, A.T. Campbell, and L. Hauw, "A Survey of QoS Architectures," ACM Multimedia Systems J., special issue on QoS architecture, vol. 6, no. 3, pp. 138-151, May 1998.

[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," RFC 2475, Dec.1998.

[3] A. Campbell, G. Coulson, and D. Hutchison, "A Quality of Service Architecture," ACM Computer Comm. Rev., vol. 24, no. 2, pp. 6-27, Apr. 1994.

[4] A. Casimiro and P. Veri'ssimo, "Using the Timely Computing Base for Dependable QoS Adaptation," Proc. 20th Symp. Reliable Distributed Systems (SRDS), pp. 208-217, Oct. 2001.



[5] T.D. Chandra, V. Hadzilacos, and S. Toueg, "The Weakest Failure Detector for Solving Consensus," J. ACM, vol. 43, no. 4, pp. 685-722, July 1996.

### Sites Referred:

<http://java.sun.com>

<http://www.sourceforge.com>

<http://www.jfree.org/>

<http://www.networkcomputing.com/>

### Sample Screen:

Screen 1:



The screenshot shows a Java Swing window titled "Fault\_Client1 - extends JFrame". The window has a header banner with a target icon, the text "Fault Tolerance System For Distributed Networking", a row of green plants, and a black arrow pointing right with the text "THIS WAY". Below the banner, the interface is divided into several sections:

- Selected Destination:** A text input field.
- Enter the Text:** A larger text input field.
- (or)**: A label indicating an alternative input method.
- Selected File:** A text input field.
- Browse**: A button next to the "Selected File" field.
- Status:** A section with three radio buttons: "Live" (selected), "Uncertain", and "Down".
- Destinations:** A list box containing the text "bct-17".
- Buttons:** At the bottom, there are two buttons: "Send" and "Logout".



**Fault\_Client1 - extends JFrame**

## Fault Tolerance System For Distributed Networking

**THIS WAY** →

**Selected Destination:**

**Enter the Text:**

(or)

**Selected File:**  **Browse**

**Destinations**

bct-17

**Status:** ☒ Live ☐ Uncertain ☐ Down

**Send** **Logout**

**SQL Query Analyzer**

File Edit Query Tools Window Help

Query - BCT-17.fault.BCT-17\admin - Untitled1\*

```
select * from userlist
```

	username	state
1	bct-17	Live

Query batch completed.

BCT-17 (8.0) BCT-17\admin (54) fault 0:00:00 1 rows Ln 1, Col 1

Connections: 1 NUM

**Fault\_Client1 - extends JFrame**

## Fault Tolerance System For Distributed Networking

**THIS WAY** →

**Selected Destination:**

**Enter the Text:**

(or)

**Selected File:**  **Browse**

**Destinations**

**Status:** ☐ Live ☒ **Uncertain** ☐ Down

**Send** **Logout**

**SQL Query Analyzer**

File Edit Query Tools Window Help

Query - BCT-17.fault.BCT-17\admin - Untitled1\*

```
select * from userlist
```


	username	state
1	bct-17	Uncertain

Query batch completed.



BCT-17 (8.0) BCT-17\admin (54) fault 0:00:00 1 rows Ln 1, Col 1

Document2 - Microsoft Word Connections: 1 NUM

**Fault\_Client1 - extends JFrame**



## Fault Tolerance System For Distributed Networking

**Selected Destination:**

**Enter the Text:**

(or)

**Selected File:**

**Status:** ☐ Live ☐ Uncertain ☒ Down

**Destinations**

**SQL Query Analyzer**

File Edit Query Tools Window Help

Query - BCT-17.fault.BCT-17\admin - Untitled1\*

```
select * from userlist
```

	username	state
1	bct-17	Down

Query batch completed.

BCT-17 (8.0) BCT-17\admin (54) fault 0:00:00 1 rows Ln 1, Col 1

Connections: 1 NUM

## Sample Code:

```

/*****
/*          Fault_Client1          */
/*
/*
*****/

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.net.*;
import java.util.*;
import java.io.*;
import java.sql.*;
import java.awt.Toolkit;
import java.util.Timer;
import java.util.TimerTask;
/**
 * Summary description for Fault_Client1
 *
 */
public class Fault_Client extends JFrame implements ItemListener
{
    // Variables declaration
    //private JTabbedPane jTabbedPane1;
    private JPanel contentPane;
    //-----
    private JTextField jTextField1;
    private JList jList1;
    private JScrollPane jScrollPane2;
    private JTextArea jTextArea2;
    private JScrollPane jScrollPane3;
    //private JPanel jPanel1;
    //-----
    private JTextArea jTextArea1;

```

```

private JScrollPane jScrollPane1;
private JPanel jPanel2;
private JRadioButton jButton1;
private JRadioButton jButton2;
private JRadioButton jButton3;
private JTextField jTextField2;
private JButton jButton1;
private JButton jButton2;
private JButton jButton3;
private ButtonGroup btnGroup1;
private JLabel jLabel1;
private JLabel jLabel2;
private JLabel jLabel3;
private JLabel jLabel4;
private JLabel jLabel5;
private JLabel jLabel6;
private JLabel jLabel7;
Toolkit toolkit;
Timer timer;
boolean sss;
int numWarningBeeps=3 ;
FileDialog fd;
int emp;
private String ServerSystemName="bct-17";
String userStatus,host;
Vector uncertain=new Vector();
    Statement st1;

//-----
// End of variables declaration

public Fault_Client()
{

```

```

super();
initializeComponent();
//
// TODO: Add any constructor code after initializeComponent call
//

this.setVisible(true);
if(jRadioButton1.isSelected( ) )
{

    String sr1 = (jRadioButton1.getText());

    System.out.println("The Selected button is"+sr1 );
    updateDb(sr1);
}

//login();
recieveMessage();

//      try
//      {
//          ss = new ServerSocket(9396);
//      }
//      catch (Exception ee)
//      {
//          ee.printStackTrace();
//      }

/*jTextArea1.setText("Destination Started ....");
toolkit = Toolkit.getDefaultToolkit();
timer = new Timer();
timer.scheduleAtFixedRate(new RemindTask(), 0, 1 * 1000);*/

```



```

    }

    public void itemStateChanged(ItemEvent e)
    {
    }

    public void login()
    {
        try
        {
            System.out.println("\n\n\n----->    Inside    the    Login
Function...<-----\n\n\n");

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

            Connection
con1=DriverManager.getConnection("jdbc:odbc:fault","sa","");

            Statement
st1=con1.createStatement();

            String app="SELECT * FROM userlist";
            ResultSet rs=st1.executeQuery(app);
            InetAddress inet=InetAddress.getLocalHost();
            String sname=inet.getHostName();//f.toUpperCase();
            while(rs.next())
            {
                if(rs.getString(1).equals(sname))
                {
                    JOptionPane.showMessageDialog(Fault1.this," Already Source is running ... ",
                    "Information          :          Click          Ok          Button          To
Exit...",JOptionPane.INFORMATION_MESSAGE);

                    //

                    Thread.sleep(1000);
                }
            }
        }
    }
}

```

```

//      System.exit(0);
    }
}

```

```

String query = "insert into userlist values('"+sname+"','live')";
st1.executeUpdate(query);

```

```

    }

```

```

catch (Exception eee)

```

```

    {

```

```

        System.out.println(eee);

```

```

    }

```

```

}

```

```

//// Here the timer task function */

```

```

//      class RemindTask extends TimerTask

```

```

// {

```

```

//      public void run()

```

```

//      {

```

```

//          while (true)

```

```

//          {

```

```

//

```

```

//          sss=true;

```

```

//          while (sss)

```

```

//          {

```

```

//              if (numWarningBeeps > 0)

```

```

//              {

```

```

//                  long time = System.currentTimeMillis();

```

```

//                  if (time - scheduledExecutionTime() > 5)

```

```

//                      {

```

```

//          return;
//      }
//      //If it's not too late, beep.
//      // toolkit.beep();
//
//      System.out.println("Updating the source Details!" + num Warning Beeps);
//          userupdate();
//          recieveMessage();
//          numWarningBeeps--;
//      }
//      else
//      {
//          // toolkit.beep();
//          System.out.println("Time's up!");
//          sss=false;
//          numWarningBeeps = 2;
//          userupdate();
//          recieveMessage();
//          break;
//          //timer.cancel(); //Not necessary because we call System.exit
//          //System.exit(0); //Stops the AWT thread (and everything else)
//      }
//
//      } // if closed
//  } // run() closed
//
// }
// }

```

```

public void userupdate()

```

```

{

```

```

    Vector ft=new Vector();

```

```

try
{
    //System.out.println(" Inside the user update .....");
    Connection con1=null;
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    con1=DriverManager.getConnection("jdbc:odbc:fault","sa","");
    Statement st1=con1.createStatement();
    String cou="SELECT count(*) FROM userlist";
    ResultSet rs=st1.executeQuery(cou);
    int count=0;
    while(rs.next())
    {
        count=rs.getInt(1);
    }
    //System.out.println("\n\n count : "+count);
    String app="SELECT * FROM userlist";
    rs=st1.executeQuery(app);

    while(rs.next())
    {
        //System.out.println(" Inside the TABLE.....");
        String name=rs.getString(1);
        ft.add(name);
        //System.out.println(ft);
    }

    //Thread.sleep(10000);

}

```

```

        catch (Exception eee)
        {
            System.out.println(" Could not get The table values...."+eee);
        }

        jList1.setListData(ft);

                                                //
    }

    private void initializeComponent()
    {
        fd=new FileDialog(this," Select the File",FileDialog.LOAD);
        fd.setFile("*.txt");
        //jTabbedPane1 = new JTabbedPane();
        contentPane = (JPanel)this.getContentPane();
        //-----
        jTextField1 = new JTextField();
        jList1 = new JList();
        jScrollPane2 = new JScrollPane();
        jTextArea2 = new JTextArea();
        jScrollPane3 = new JScrollPane();
        //jPanel1 = new JPanel();
        //-----
        jTextArea1 = new JTextArea();
        jScrollPane1 = new JScrollPane();
        jRadioButton1 = new JRadioButton();
        jRadioButton2 = new JRadioButton();
        jRadioButton3 = new JRadioButton();
        jTextField2 = new JTextField();
        jButton1 = new JButton();
        jButton2 = new JButton();
        jButton3 = new JButton();
        btnGroup1 = new ButtonGroup();
        jLabel1 = new JLabel();
        jLabel2 = new JLabel();
        jLabel3 = new JLabel();
    }

```

```

jLabel4 = new JLabel();
jLabel5 = new JLabel();
jLabel6 = new JLabel();
jLabel7 = new JLabel(new ImageIcon("OCGRR.jpg"));
jPanel2 = new JPanel();
setDefaultCloseOperation(EXIT_ON_CLOSE);
//-----// jTabbedPane1

/*jTabbedPane1.addTab(" Send ", jPanel1);
//jTabbedPane1.addTab(" Recieve ", jPanel2);
jTabbedPane1.addChangeListener(new ChangeListener() {
    public void stateChanged(ChangeEvent e)
    {
        jTabbedPane1_stateChanged(e);
    }
});*/
//
// contentPane
//
contentPane.setLayout(null);
//addComponent(content Panej TabbedPane1,
13,55,548,450);//13,55,548,391;
//
// jTextField1
//
jLabel1.setText("Selected Destination:");
jLabel2.setText("Enter the Text:");
jLabel3.setText("(or)");
jLabel4.setText("Selected File:");
jLabel5.setText("Destinations");
jLabel6.setText("Status:");
jTextField1.setText("");

```

```

jTextField1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e)
    {
        jTextField1_actionPerformed(e);
    }

});
//
// jList1
//
jList1.addListSelectionListener(new ListSelectionListener() {
    public void valueChanged(ListSelectionEvent e)
    {
        jList1_valueChanged(e);
    }

});
//
// jScrollPane2
//
jScrollPane2.setViewportViewView(jList1);
//
// jTextArea2
//
jTextArea2.setText("");
//
// jScrollPane3
//
jScrollPane3.setViewportViewView(jTextArea2);
jRadioButton1.setText(" Live");
jRadioButton1.setSelected(true);

jRadioButton1.addItemListener(new ItemListener() {

```

```

        public void itemStateChanged(ItemEvent e)
        {
            jRadioButton1_itemStateChanged(e);
        }

    });
    //
    // jRadioButton2
    //
    jRadioButton2.setText(" Uncertain");
    jRadioButton2.addItemListener(new ItemListener() {
        public void itemStateChanged(ItemEvent e)
        {
            jRadioButton2_itemStateChanged(e);
        }

    });
    //
    // jRadioButton3
    //
    jRadioButton3.setText(" Down");
    jRadioButton3.addItemListener(new ItemListener() {
        public void itemStateChanged(ItemEvent e)
        {
            try
            {

                jRadioButton3_itemStateChanged(e);
                Socket sock;
                ServerSocket ss;
                Socket s2,s1;

```



```

        ObjectOutputStream ois;
        ObjectOutputStream oos=null;

        /*sock =new Socket(ServerSystemName,9397);
        ois = new
ObjectInputStream(sock.getInputStream());
        oos = new      ObjectOutputStream
(sock.getOutputStream());

        oos.writeObject("hai");
        */
    }
    catch (Exception rt)
    {
        rt.printStackTrace();
    }

}

});
//
// jTextField2
//
jTextField2.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e)
    {

        //jTextField2_actionPerformed(e);
    }

});
//
// jButton1
//

```

```

jButton1.setText(" Browse");
jButton1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e)
    {
        fd.setVisible(true);
        String filename = fd.getDirectory()+fd.getFile();
        if(filename.equals("nullnullnull"))
        {
            jTextField2.setText("");
        }
        else if(filename.equals("nullnull"))
        {
            jTextField2.setText("");
        }
        else if(filename.equals("null"))
        {
            jTextField2.setText("");
        }
        else
        {
            jTextField2.setText(filename);
        }
        System.out.println(" file : "+filename);
    }

});
//
// jButton2
//
jButton2.setText(" Send");
jButton2.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e)
    {
        try

```

```

        {

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

            Connection
con1=DriverManager.getConnection("jdbc:odbc:fault","sa","");

            st1=con1.createStatement();
            InetAddress inet=InetAddress.getLocalHost();
            host=inet.getHostName();
            String app="SELECT state FROM userlist
where username like '%" + host + "%' ";

            ResultSet rs=st1.executeQuery(app);
            while(rs.next())
            {
                userStatus = rs.getString(1);
            }
            userStatus=userStatus.trim();
            System.out.println("---->      System      Status

:" + app);

            System.out.println("---->      System      Status

:" + userStatus);

        }
        catch (Exception excep)
        {
            System.out.println(excep);
        }

        //jButton2_actionPerformed(e);

        int uy=jTextField2.getText().length();
        String ccd = "Destination is Down state.Please try after
some time";

        if(uy==0)

```

```

    {

if(userStatus.equals("Live"))
    {
        sendMessage();
    }
else if(userStatus.equals("Uncertain"))
    {
        String dest = jTextField1.getText().trim();
        System.out.println("The destination is" + dest);
        String s12 = jTextArea2.getText();
        System.out.println("The Text Message is" + s12);
        //InetAddress inet=InetAddress.getLocalHost();
        //String host=inet.getHostName();
        String message = host+"$"+dest+"$"+s12;
        uncertain.add(message);

    }
else if (userStatus.equals("Down"))

        {
            JOptionPane.showMessageDialog(Fault_Client.this,ccd,
            "Message ...",JOptionPane.INFORMATION_MESSAGE);
        }
        System.out.println("  Uncertain  Vector
Values : "+uncertain);

        System.out.println("Inside The Directly
Msg Send..Not Browse Selection..");
    }
    else
    {
        jTextArea2.setText("");
        String filen=jTextField2.getText();
        try
        {

```

```

BufferedReader      in=new BufferedReader(new FileReader(filen));

                    String s,s1=new String();
                    if(jTextArea2.getText().length()==0)
                    {
                        emp=1;
                    }
                    while((s=in.readLine())!=null)
                    {
                        if(emp==0)
                        {

jTextArea2.setText(jTextArea2.getText()+"\n"+s);

                        //emp=1;
                        }
                        else
                        {

                            jTextArea2.setText(s);
                            emp=0;
                        }

                    }
                    if(userStatus.equals("Live"))
                    {

                        sendMessage();
                    }
                    else if(userStatus.equals("Uncertain"))
                    {

String dest = jTextField1.getText().trim();
System.out.println("The destination is" + dest);
String s12 = jTextArea2.getText();
System.out.println("The Text Message is" + s12);

```

```

//InetAddress
inet=InetAddress.getLocalHost();

//String
host=inet.getHostName();
String message = host+"$"+dest+"$"+s12;

        uncertain.add(message);
    }

    else if(userStatus.equals("Down"))
OptionPane.showMessageDialog(Fault_Client.this,ccd,                "Message
...",JOptionPane.INFORMATION_MESSAGE);
    }

        //jTextField1.setText("");
    }
    catch (Exception er)
    {
        System.out.println(er);
    }

//System.out.println("Nooooooooooooooooooooooooooooo..");

    }

}

});
//
// jButton3
//

```

```

jButton3.setText(" Logout");
jButton3.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e)
    {
        //jButton3_actionPerformed(e);
    }

});
//
// jPanel1
//
//
// jPanel1
//
//jPanel1.setLayout(null);
//jPanel1.setBorder(BorderFactory.createEtchedBorder());
addComponent(contentPane, jTextField1, 173,130,132,28);
addComponent(contentPane, jScrollPane2, 360,140,100,126);
addComponent(contentPane, jScrollPane3, 155,190,172,67);
addComponent(contentPane, jButton1, 158,350,56,24);
addComponent(contentPane, jButton2, 210,350,84,24);
addComponent(contentPane, jButton3, 298,350,65,24);
addComponent(contentPane, jTextField2, 154,300,172,25);
addComponent(contentPane, jButton1, 353,300,83,28);
addComponent(contentPane, jButton2, 158,395,83,28);
addComponent(contentPane, jButton3, 254,395,83,28);
addComponent(contentPane, jLabel1, 25,130,122,28);
addComponent(contentPane, jLabel2, 61,209,84,32);
addComponent(contentPane, jLabel3, 215,268,72,20);
addComponent(contentPane, jLabel4, 67,300,84,20);
addComponent(contentPane, jLabel5, 369,110,84,20);
addComponent(contentPane, jLabel6, 87,350,84,20);
addComponent(contentPane, jLabel7, 0,0,490,100);
//

```

```

// jPanel2
//
jPanel2.setLayout(null);
jPanel2.setBorder(BorderFactory.createEtchedBorder());
//
// btnGroup1
//
btnGroup1.add(jRadioButton1);
btnGroup1.add(jRadioButton2);
btnGroup1.add(jRadioButton3);
//
//
// jTextArea1
//
jTextArea1.setText("\n");
//
// jScrollPane1
//
jScrollPane1.setViewportView(jTextArea1);
//
// jPanel2
//
jPanel2.setLayout(null);
jPanel2.setBorder(BorderFactory.createEtchedBorder());
addComponent(jPanel2, jScrollPane1, -1,0,548,363);
MouseListener mouseListener = new MouseAdapter()
{
    public void mouseClicked(MouseEvent mouseEvent)
    {
        JList jList1 = (JList) mouseEvent.getSource();
        if (mouseEvent.getClickCount() == 2)
        {
            int index = jList1.locationToIndex(mouseEvent.getPoint());
            if (index >= 0)

```



```

{
Object o = jList1.getModel().getElementAt(index);

//System.out.println(">>" + ((o==null)? "null" : o.toString()) + " is selected.");
//String tatest=jTextArea1.getText();
jTextField1.setText(o.toString());
//view v1=new view(o.toString());

//System.out.println("Double-clicked on: " + o.toString());
}

}

}

};
jList1.addMouseListener(mouseListener);

//
// Fault_Client1
//
this.setTitle("Fault_Client1 - extends JFrame");
this.setLocation(new Point(0, 0));
this.setSize(new Dimension(485, 485));// 580,506
}

private void jRadioButton1_itemStateChanged(ItemEvent e)
{
    if(jRadioButton1.isSelected())
    {
        String sr1 = (jRadioButton1.getText());
        System.out.println("The Selected button is"+sr1 );
        updateDb(sr1);
    }
}

```

```

private void jButton2_itemStateChanged(ItemEvent e)
{
    if(jButton2.isSelected())
    {
        String sr1 = (jButton2.getText());
        System.out.println("The Selected button is"+sr1 );
        updateDb(sr1);
    }

}

```

```

private void jButton3_itemStateChanged(ItemEvent e)
{
    if(jButton3.isSelected())
    {
        String sr1 = (jButton3.getText());
        System.out.println("The Selected button is"+sr1 );
        updateDb(sr1);
    }

}

```

```

/*public void itemStateChanged(ItemEvent ie)

```

```

{

    if(jr2.isSelected( ))
    {
        System.out.println("Uncertain");
        String sr1 = (jr2.getText());
        //updateDb(sr1);
    }
    else if(jr3.isSelected( ))

```

```

        {
            System.out.println("Down");
            String sr1 = (jr3.getText());
            //updateDb(sr1);
        }
        else if( jr1.isSelected( ) )
        {
            System.out.println("Live" );
            String sr1 = (jr1.getText());

            //updateDb(sr1);
        }

    }*/

    /** 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 jTablebedPane1_stateChanged(ChangeEvent e)
    {
        System.out.println("\njTablebedPane1_stateChanged(ChangeEvent    e)
called.");
    }

```

```

        // TODO: Add any handling code here

    }

    private void jTextField1_actionPerformed(ActionEvent e)
    {
        System.out.println("\njTextField1_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.");
            //      jTextField1.setText(o.toString());
            // TODO: Add any handling code here for the particular object
being selected

        }
    }

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

    public void updateDb(String sr1)

```

```

{
    String sr = sr1;
    System.out.println("The Selected Radio button is"+sr);

    try
    {
        InetAddress inet=InetAddress.getLocalHost();
        String sname=inet.getHostName();
        System.out.println(" Inside the user update .....");
        Connection con1=null;
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        con1=DriverManager.getConnection("jdbc:odbc:fault","sa","");
        Statement st1=con1.createStatement();
        String query = "update userlist set state = '"+sr+"' where
username like '%" +sname+"%' ";

        st1.executeUpdate(query);

    }
    catch(Exception e)
    {

        e.printStackTrace();

    }
}

//private void jButton2_actionPerformed(ActionEvent e)
//    {
//
//        sendMessage();
//
//    }

```

```

public void sendMessage()
{
    Socket sock;
    ServerSocket ss;
    Socket s2,s1;
    ObjectInputStream ois;
    ObjectOutputStream oos=null;
    String msg = "";
    String ccd = "The message is recieved";

    System.out.println("-----Inside the sendMessage function--
-----");
    try
    {

        System.out.println("--> Uncertain size : "+uncertain.size());
        if(uncertain.size()>0)
        {
            for(int count=0;count<uncertain.size();count++)
            {
                String val1=(String)uncertain.elementAt(count);
                System.out.println(" value "+count + " : "+val1);
                try
                {
                    sock =new Socket(ServerSystemName,9395);
                    ois = new ObjectInputStream
ObjectInputStream(sock.getInputStream());
                    oos = new ObjectOutputStream
(sock.getOutputStream());
                    oos.writeObject(val1);

```

```

        msg = (String)ois.readObject();
        System.out.println("THE MESSAGE IS
RECIEVED"+msg);

        if(msg.equals("Recieved"))
        {

            JOptionPane.showMessageDialog(Fault_Client.this,ccd, "Message
...",JOptionPane.INFORMATION_MESSAGE);

        }
    }
    catch (Exception ex)
    {

        ex.printStackTrace();

    }

}

uncertain.removeAllElements();
}
}
catch (Exception exc)
{
}
//else
//{

String dest = jTextField1.getText().trim();
System.out.println("The destination is" + dest);
String s = jTextArea2.getText();

System.out.println("The Text Message is" + s);
try
{

    InetAddress

inet=InetAddress.getLocalHost();

    String host=inet.getHostName();

```

```

        String message = host+"$"+dest+"$"+s;

        sock = new
Socket(ServerSystemName,9395);

        ois = new
ObjectInputStream(sock.getInputStream());

        oos = new ObjectOutputStream
(sock.getOutputStream());

        oos.writeObject(message);

        //String msg =
(String)ois.readObject();

        msg = (String)ois.readObject();
        System.out.println("THE MESSAGE IS
RECIEVED"+msg);

        if(msg.equals("Recieved"))
        {

            JOptionPane.showMessageDialog(Fault_Client.this,ccd, "Message
...",JOptionPane.INFORMATION_MESSAGE);

        }

    }
    catch (Exception e)
    {

        e.printStackTrace();

    }

    //}

}

```



```

public void recieveMessage()
{

    try
    {
        //ss = new ServerSocket(9396);

        System.out.println("Destination started...");
        while(true)
        {
            userupdate();
            Thread.sleep(1000);
        }

    }
    catch (Exception e)
    {
        e.printStackTrace();
    }

}

```

## 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("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
//        }
//        catch (Exception ex)
//        {
//            System.out.println("Failed loading L&F: ");
//            System.out.println(ex);
//        }
//    }
}
```

```
//      }  
      new Fault_Client();  
  }  
  // = End of Testing =  
  
}
```