1. INTRODUCTION

1.1 Introduction:

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 non crashed 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 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. 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.

2. FEASIBILITY STUDY

A feasibility study is a test of the system proposal regarding to its work ability, impact on the organization, ability to meet user needs, and effective use of resources. Thus when a new application is proposed, it normally goes through a feasibility study before it's is approved for development. An important outcome of the preliminary investigation is the determination that the system requested is feasible. The feasibility study was conducted under these aspects:

2.1 Technical Feasibility:

Technical feasibility deals with hardware as well as software requirement. In the feasibility study scope was whether the work for the project could be done with the current equipment and the existing software technology. The outcome of the study was positive. It was found that all software and hardware specification were available.

2.2 Operational Feasibility:

The purpose of the operational feasibility study was to determine whether the new system would be used if it is developed and implemented. Will there be resistance from the users that will undermine the possible application benefits? From the output of the meeting that was help with the system user, it was found that all of them supported the development of the new system. The positive response from them encouraged us in the development of the new system.

2.3 Economic Feasibility:

Economic feasibility is the most frequently used method for evaluating the effectiveness of a candidates system. More commonly known as cost/benefit analysis, the procedure is to determine the benefits and savings that are accepted from a candidate system and compare them with costs. If benefits outweigh costs, further

justification and alterations in the proposed system will have to be made if it is to have a chance of begin approved.

The economic feasibility study was feasible so that the project could be undertaken. Another question was whether the proposed system would be beneficial to make it cost effective. This project will provide the organization with saving of time and saving in employee salary. Here it was found that benefits were more than the costs and thus the project passed economic feasibility.

3. REQUIREMENTS ELICITATION AND ANALYSIS DOCUMENT

3.1 Introduction:

3.1.1 Purpose:

The main purpose of the System is to improve the system capacity and reduce the handoff delay for wireless Multicast and Broadcast Service (MBS), the MBS zone technology is being proposed in several Mobile Communications Network (MCN). Multimedia Broadcast multicast service Single Frequency Network Area in 3GPP Universal Mobile Telecommunications System (UMTS) and Long Term Evolution (LTE). The MBS Controller (MBSC; accommodates the functionalities including the MBS zone management, service announcement, membership management, security management, session management, session transmission, multicast connection identifier and IP address management.

3.1.2 Scope:

This project plays a vital role in three development life cycle (SDLC) as it describes the complete requirement of the system. It is meant for use by the developers and will be the basic during testing phase. Any changes made to the requirement in the future will have to go through the changes approval process.

The developers are responsible for:

- Developing the system, which meets the SRS and solving all the requirements of the system?
- Demonstrating the system and installing the system at client's location after the acceptance testing is successful.
- Submitting the required user manual describing the system interfaces to work on it and also the documents of the system.
- Conducting all users training that might be needed for using the system.
- Maintaining the system for a period of one year after installation

3.1.3 Objectives and Successive Criteria:

This document plays a vital role in the system development life cycle as it describes the complete requirements of the system. It is meant for use by the developers and will be the basic during the testing phase. Any changes made to the requirements in the future will have to go through formal change approval process.

The main objectives are as follows-

1. No delay in transferring of messages.
2. It increases the system capacity for MBS content.
3. Low faults and failures of the system.
4. This paper proposed and fully developed an adaptive programming model for fault-tolerant distributed computing.
5. Our model provides upper-layer applications with process state information according to the current system synchrony (or QoS).

3.2 Current 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 non crashed 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).

Disadvantages:

1. The existing system is usually called a time-free asynchronous distributed system prone to process crashes.
2. In these systems, a system designer can only assume an upper bound on the number of processes that can crash and, consequently, design protocols relying on the assumption that at least processes are alive. The protocol has no means to know whether a given process is alive or not.

3.3 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 Pd (deadline probability) and Pr (reliability probability) close to one, and untimely channels could be implemented with a basic channel without any guarantees.

Advantages:

1. Our model provides upper-layer applications with process state information according to the current system synchrony (or QoS).
2. The underlying system model is hybrid, comprised of a synchronous part and an asynchronous part.

3.4 Functional Requirements:

The functional specification is designed to be read by a general audience. Readers should understand the system, but no particular technical knowledge should be required to understand the document.

Functional Requirements should include:

- Descriptions of data to be entered into the system
- Descriptions of operations performed by each screen
- Descriptions of work-flows performed by the system
- Descriptions of system reports or other outputs
- Who can enter the data into the system?
- How the system meets applicable regulatory requirements.

Examples of Functional Requirements:

Functional requirements should include functions performed by specific screens, outlines of work-flows performed by the system and other requirements the system must meet.

- **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 and 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.

3.5 Non – Functional Requirements:

Non-functional requirements describe user-visible aspects of the system that are not directly related to functionality of the system.

i. User Interface:

A menu interface has been provided to the client to be user friendly.

ii. Documentation:

The client is provided with an introductory help about the client interface and the user documentation has been developed through help hyperlink.

iii. Performance Constraints, Reliability:

- Requests should be processed within no time.
- Users should be authenticated for accessing the requested data

iv. Error Handling and Extreme Conditions:

In case of User Error, the System should display a meaningful error message to the user, such that the user can correct his Error. The high level components in proposed system should handle exceptions that occur while connecting to database server, IOExceptions etc.

v. Quality Issues:

Quality issues refer to how reliable, available and robust should the system be? While developing the proposed system the developer must be able to guarantee the reliability transactions so that they will be processed completely and accurately.

The **ability** of system to detect failures and recovery from those failures refers to the availability of system.

Robustness of system refers to the capability of system providing information when concurrent users requesting for information.

vi. Acceptance Criteria:

The developer will have to demonstrate and show to the user that the system works by testing with suitable test cases so that all conditions are satisfied.

3.6 Pseudo Requirements:

3.6.1 Software Requirements:

- Operating system : Windows XP Professional
- Frontend : Java, Swings
- Tool kit : JDK 1.5 or above, JFrame Builder

3.6.2 Hardware Requirements:

- System : Pentium IV 2.4 GHz
- Hard Disk : 40 GB
- Floppy Drive : 1.44 MB
- Monitor : 15 VGA colour
- RAM : 256 MB
- Input devices : Standard Keyboard and Mouse.

3.6.3 Software Overview:

Java Technology:

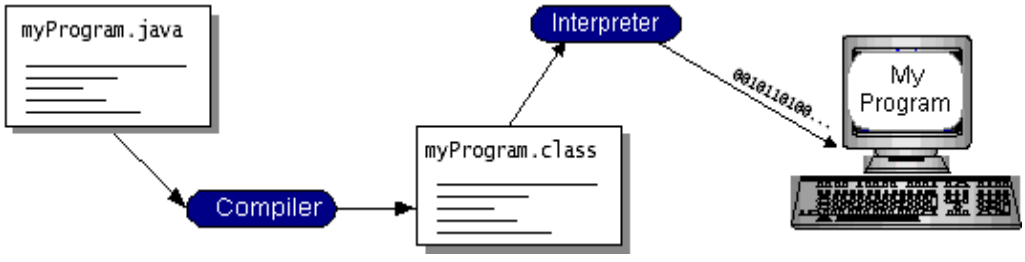
Java technology is both a programming language and a platform.

The Java Programming Language

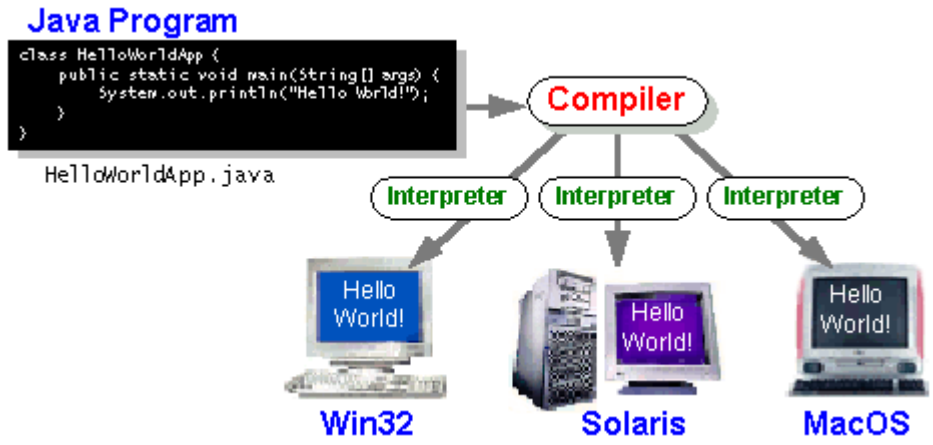
The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

- 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 byte codes 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 byte codes help make "write once, run anywhere" possible. You can compile your program into byte codes on any platform that has a Java compiler. The byte codes 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.

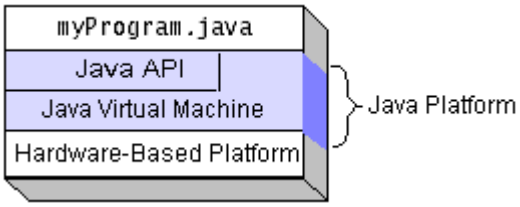
The Java platform has two components:

- 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 byte code 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.

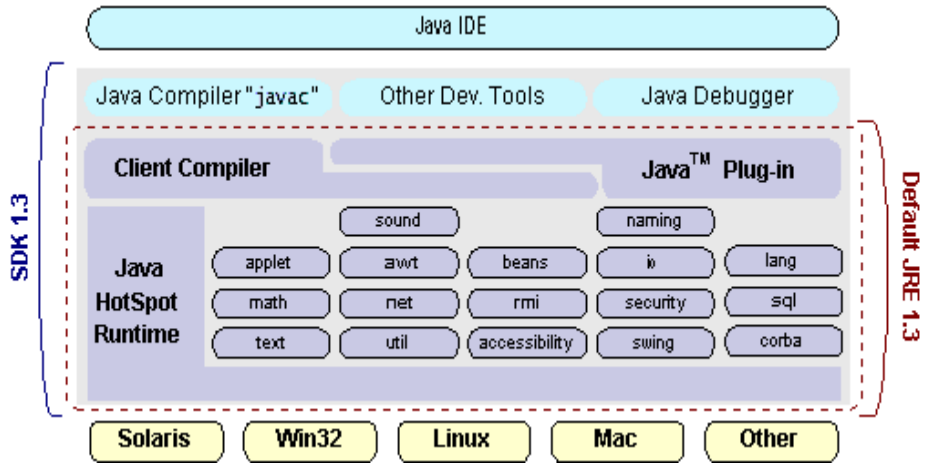
An application is a standalone program that runs directly on the Java platform. A special kind of application known as a *server* serves and supports clients on a network. Examples of servers are Web servers, proxy servers, mail servers, and print servers. Another specialized program is a *servlet*. 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 those 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 Data gram 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™, can plug into existing component architectures.
- **Object serialization**: Allows lightweight persistence and communication via Remote Method Invocation (RMI).
- **Java Database Connectivity (JDBC™)**: 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 datasources. 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. 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 error appear at runtime.

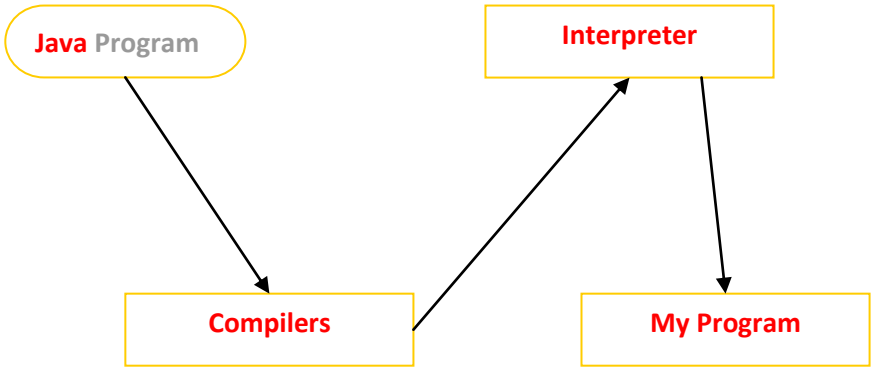
7. **Keep the common cases simple**

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 precede 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	Robust
Object-oriented	Portable	Dynamic
Distributed	High-performance	Secure
Interpreted	multithreaded	

Java is also unusual in that each Java program is both compiled and interpreted. With a compile 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.

Features of JAVA:

Platform – Independent:

Changes and upgrades in operating systems, processors and system resources will not force any change in java programs. This is the reason why Java has become a popular language for programming on Internet.

Portable:

Java ensures portability in two ways. First, java compiler generates bytecode instructions that can be implemented on any machine. Secondly, the size of the primitive data types is machine independent.

Object oriented:

Java is a true objected oriented language. Almost everything in java is an object. All program code and data must reside within objects and classes. Java comes with an extensive set of classes arranged in packages that we can use in out programs by inheritance. The object model in java is simple and easy to extend.

Distributed:

Java is designed as a distributed language for creating applications on networks. It has the ability to share both date and programs.

Dynamic:

Java is a dynamic language. Java is capable of dynamically linking new class, libraries, methods and objects.

Secure:

Since java supports applets which are programs that are transferred through internet, there may arise a security threat. But java overcomes this problem by confining the applets to the runtime package or JVM and thus it prevents infections and malicious contents.

Robust:

Java is said to be robust in two ways

1. Java allocates and de-allocates its dynamic memory on its own.
2. Java provides exception.

Multithreaded:

Java supports multithreaded programs which allow you to write programs that do many things simultaneously. This is used in interactive network programs.

Interpreted:

The byte code is interpreted by JVM. Even though interpreted, Java provides high performance. The byte code generated by the Java compiler for translating to native machine code with high performance but the Just In Time (JIT) compiler in java.

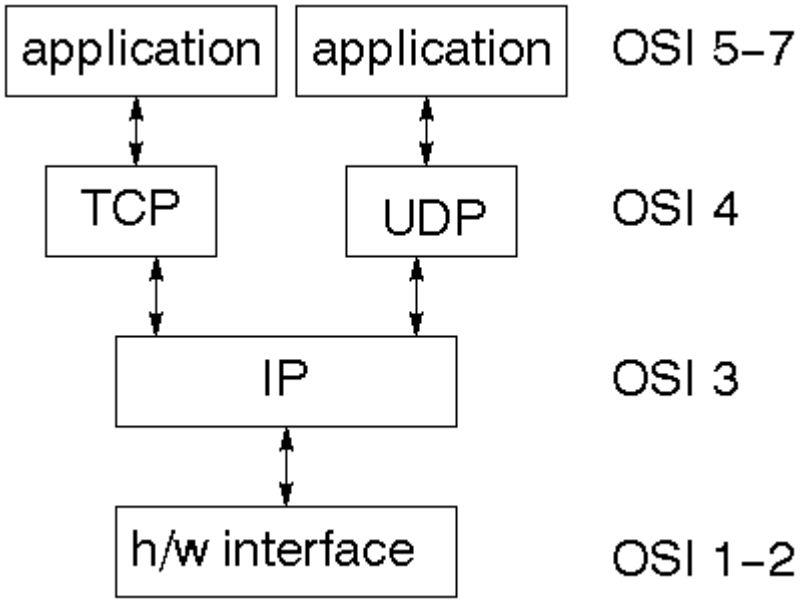
Swing	String Tokenizer
Buffered Image	Container
Image	Media Tracker
J Frame	J File Chooser

J Scroll Pane

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.

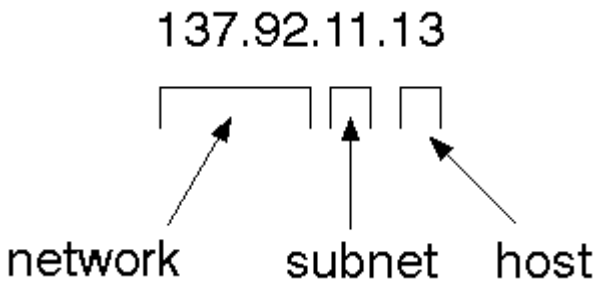
Subnet address:

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, per

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.

SWINGS:

Swing is a set of classes that provides more powerful and flexible components that are possible with AWT and hence we adapted swing. In addition to normal components such as buttons, check box, labels swing includes tabbed panes, scroll panes, trees and tables. It provides extra facilities than the normal AWT components.

J Frame: Like AWT's frame class, the JFrameclass can generate events when things happen to the window, such as the window being closed, activated, iconified or opened. These events can be sent to a window Listener if one is registered with the frame.

J File Chooser: It provides a simple mechanism for the user to choose a file. Here it points the users default directory. It includes the following methods:

Show Dialog: Pops a custom file chooser dialog with a custom approve button.

Set Dialog Type: Sets the type of this dialog. Use open-dialog when we want to bring up a file chooser that the user can use to open file. Use save-dialog for letting the user choose a file for saving.

Set Dialog Title: Set the given string as the title of the J File Chooser window.

J Scroll Pane: Encapsulates a scrollable window. It is a component that represents a rectangle area in which a component may be viewed. It provides horizontal and vertical scrollbar if necessary.

Image: The image class and the java.awt.image package, together provide the support for imaging both for the display and manipulation of web design. Images are objects

of the image class, and they are manipulated using the classes found in the java.awt.image package.

Media Tracker: Many early java developers found the image observer interface is far too difficult to understand and manage when there were multiple images to be loaded.

So the developer community was asked to provide a simpler solution that would allow programmers to load all of their images synchronously.

String Tokenizer: The processing of text often consists of parsing a formatted input string. Parsing is the division of the text in to set of discrete parts or tokens, which in a certain sequence can convey can convey a semantic meaning.

Buffered Image: The buffered Image class provides a quick, convenient shortcut by providing an image whose pixels can be manipulate directly.

3.7 System Models:

There are 3 System Models

1. Usecase or Functional Model
2. Object Model
3. Dynamic Model

3.7.1 Usecase or Functional Model:

3.7.1.1 Scenarios:

Scenario is an instance of Use case. It is a “narrative of what people do and experience as they try to make use of system and applications”. A Scenario is a concrete focused, informal description of a single feature of the system from the view point of a single actor. Scenarios cannot replace use cases, as they focus on specific instance and concrete events however, scenarios enhance requirements elicitation by providing a tool that is understandable to users and clients

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**

i. This view represents the system from the users perspective.

ii. The analysis representation describes a usage scenario from the end-users perspective.

- **Structural model view**

i. In this model the data and functionality are arrived from inside the system.

ii. 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.

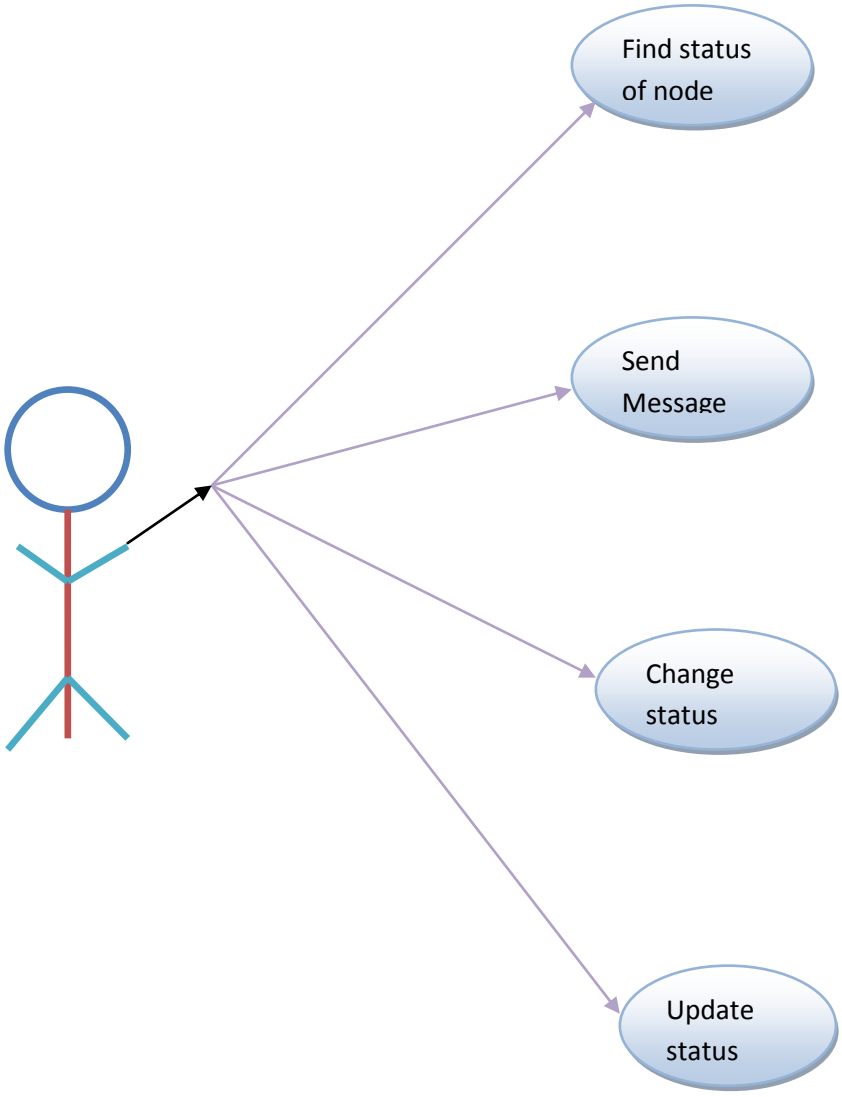
Use case Diagrams represent the functionality of the system from a user's point of view. Use cases are used during requirements elicitation and analysis to represent the functionality of the system. Use cases focus on the behavior of the system from external point of view.

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.

3.7.1.2 Usecase Diagrams:

Use Case: Use case describes the behavior of a system. It is used to structure things in a model. It contains multiple scenarios, each of which describes a sequence of actions that is clear enough for outsiders to understand.

Actor: An actor represents a coherent set of roles that users of a system play when interacting with the use cases of the system. An actor participates in use cases to accomplish an overall purpose. An actor can represent the role of a human, a device, or any other systems.



3.7.2 Object Model:

It describes the whole system using class diagrams, object diagrams. It is also describes the structure of the system.

[4]

3.7.2.1 Class Diagram:

A class is a set of objects that share a common structure and common behavior (the same attributes, operations, relationships and semantics). A class is an abstraction of real-world items.

There are 4 approaches for identifying classes:

1. Noun phrase approach:
2. Common class pattern approach.
3. Use case Driven Sequence or Collaboration approach.

4. Classes , Responsibilities and collaborators Approach

1. Noun Phrase Approach:

The guidelines for identifying the classes:

a. Look for nouns and noun phrases in the use cases.
b. Some classes are implicit or taken from general knowledge.
c. All classes must make sense in the application domain; Avoid computer implementation classes – defer them to the design stage.
d. Carefully choose and define the class names.

2. Common class pattern approach:

The following are the patterns for finding the candidate classes:

i. Concept class.
ii. Events class.
iii. Organization class
iv. Peoples class
v. Places class
vi. Tangible things and devices class.

3. Use case driven approach:

We have to draw the sequence diagram or collaboration diagram. If there is need for some classes to represent some functionality then add new classes which perform those functionalities.

4. CRC approach:

The process consists of the following steps:

a. Identify classes' responsibilities (and identify the classes)
b. Assign the responsibilities
c. Identify the collaborators.

Super-sub class relationships:

Super-sub class hierarchy is a relationship between classes where one class is the parent class of another class (derived class).This is based on inheritance.

Class:

A Class is a description for a set of objects that shares the same attributes, and has similar operations, relationships, behaviors and semantics.

Generalization:

Generalization is a relationship between a general element and a more specific kind of that element. It means that the more specific element can be used whenever the general element appears. This relation is also known as specialization or inheritance link.

Realization:

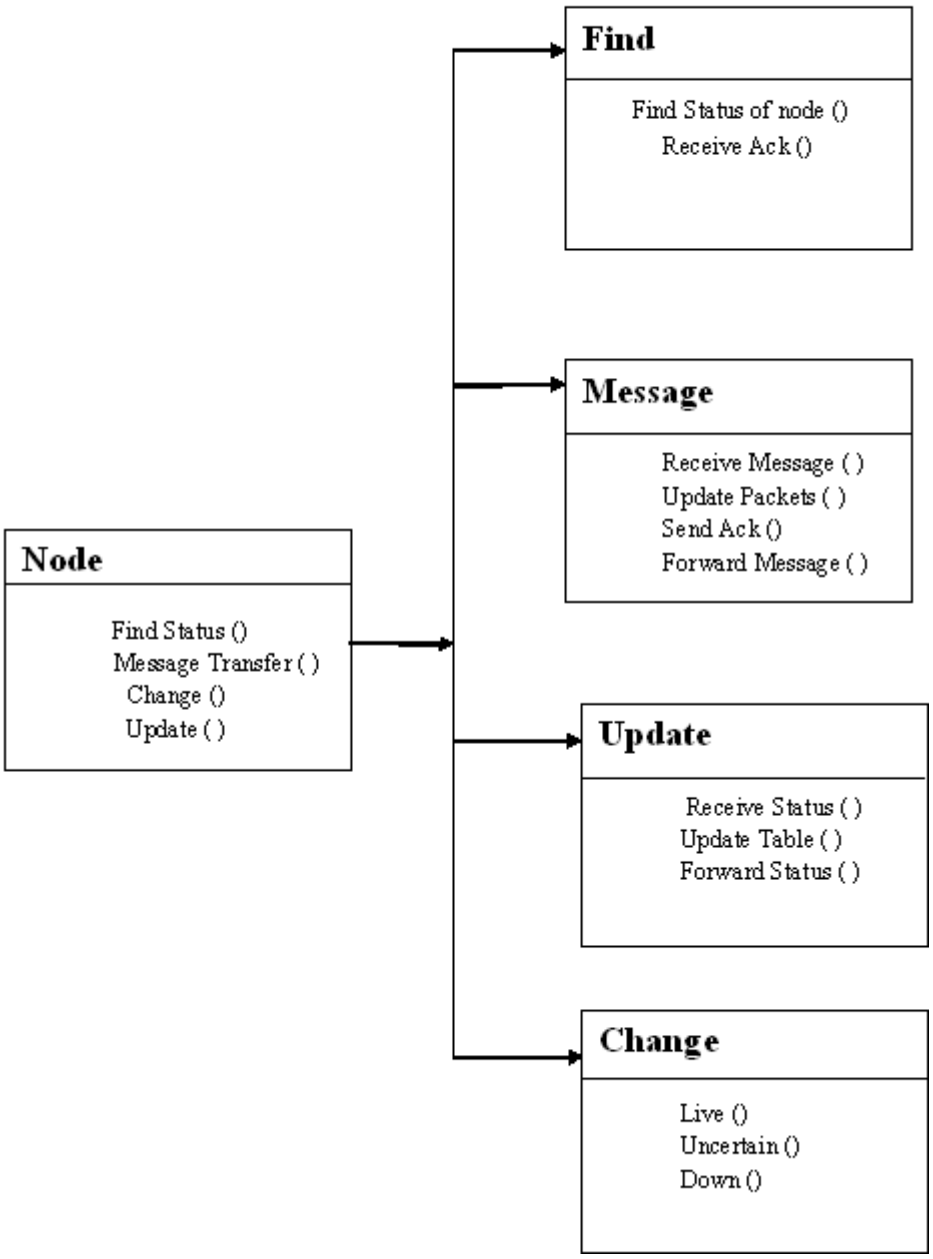
Realization is the relationship between a specialization and its implementation. It is an indication of the inheritance of behavior without the inheritance of structure.

Association:

Association is represented by drawing a line between classes. Associations represent structural relationships between classes and can be named to facilitate model understanding. If two classes are associated, you can navigate from an object of one class to an object of the class.

Aggregation:

Aggregation is a special kind of association in which one class represents as the larger class that consists of a smaller class. It has the meaning of “has-a” relationship.


Find
Find Status of node ()
Receive Ack ()
Message
Receive Message ( )
Update Packets ( )
Send Ack ()
Forward Message ( )
Node
Find Status ()
Message Transfer ( )
Change ()
Update ( )
Update
Receive Status ( )
Update Table ( )
Forward Status ( )
Change
Live ()
Uncertain ()
Down ()

3.7.3 Dynamic Model:

It describes the internal behavior of the System using sequence diagram, state chart diagram.

3.7.3.1 Sequence Diagram:

A sequence diagram is a graphical view of a scenario that shows object interaction in a time-based sequence what happens first, what happens next. Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces.

This diagram is simple and visually logical, so it is easy to see the sequence of the flow of control. It also clearly shows concurrent processes and activations in a design.

Object: Object can be viewed as an entity at a particular point in time with a specific value and as a holder of identity that has different values over time. Associations among objects are not shown. When you place an object tag in the design area, a lifeline is automatically drawn and attached to that object tag.

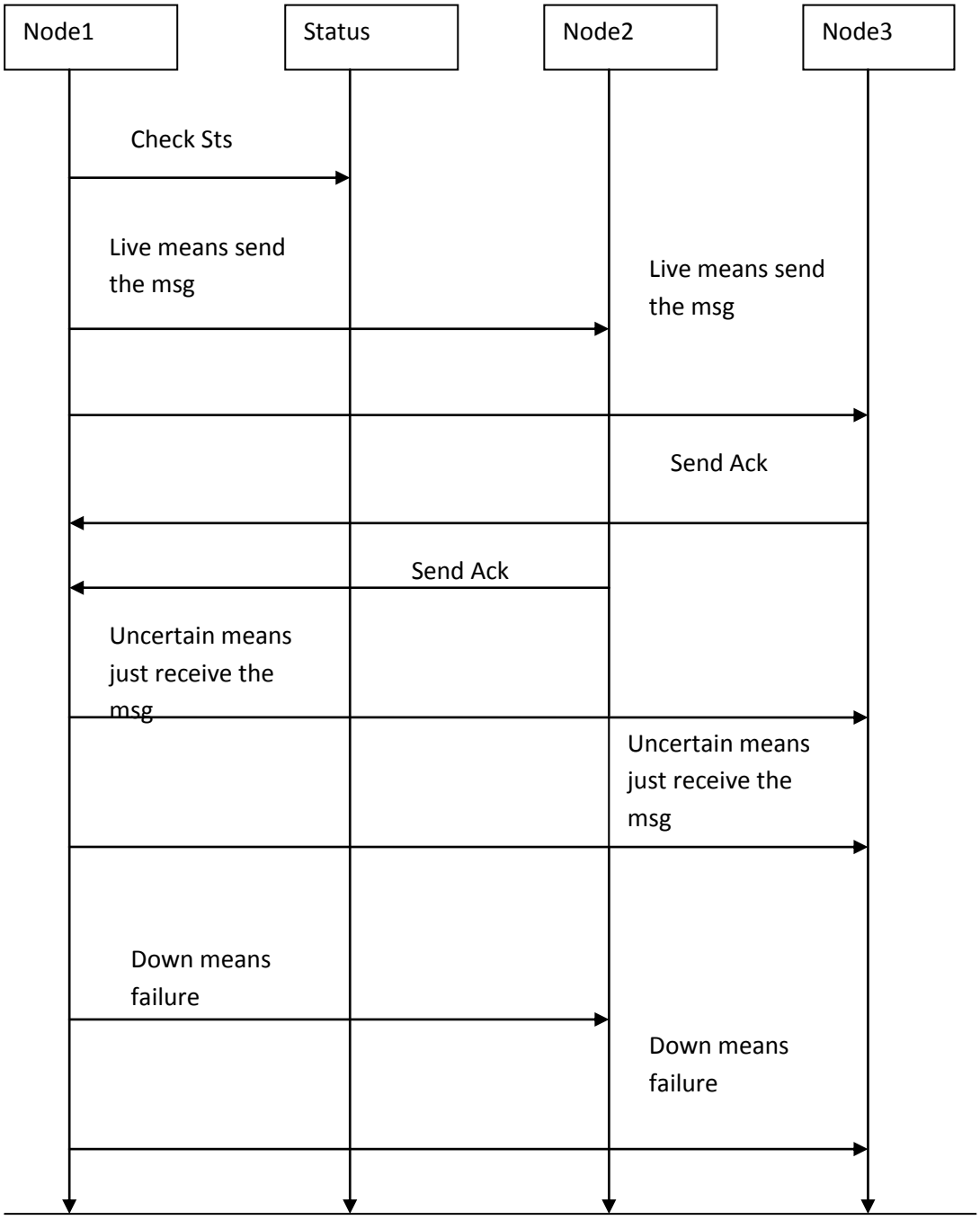
Actor: An actor represents a coherent set of roles that users of a system play when interacting with the use cases of the system. An actor participates in use cases to accomplish an overall purpose. An actor can represent the role of a human, a device, or any other systems.

Message: A message is a sending of a signal from one sender object to other receiver object(s). It can also be the call of an operation on receiver object by caller object. The arrow can be labeled with the name of the message (operation or signal) and its argument values

Duration Message: A message that indicates an action will cause transition from one state to another state.

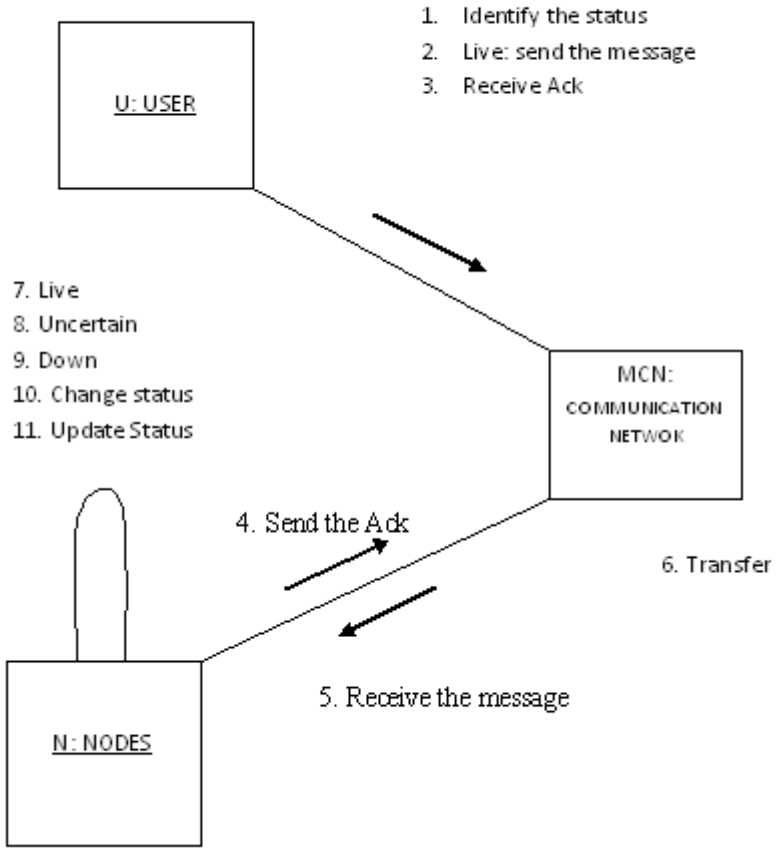
Self Message: A message that indicates an action will perform at a particular state and stay there.

Create Message: A message that indicates an action that will perform between two states.



3.7.3.2 Collaboration Diagram:

The elements of a system work together to accomplish the systems objective and a modeling language must have a way of representing this. The Uml collaboration diagram is designed for this purpose. it is an extension of the object diagram .in addition to the association among objects the collaboration diagram shows the messages the objects send each other.



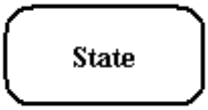
3.7.3.3 State chart Diagram:

State chart diagrams describe the behavior of an individual object as a number of states and transitions between these states. A state represents a particular set of values for an object. The sequence diagram focuses on the messages exchanged between objects, the state chart diagrams focuses on the transition between states.

A state chart diagram shows the behavior of classes in response to external stimuli. This diagram models the dynamic flow of control from state to state within a system. Basic State chart Diagram Symbols and Notations

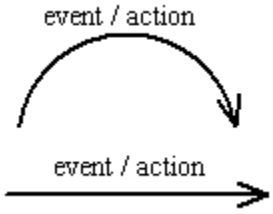
States:

States represent situations during the life of an object. You can easily illustrate a state in Smart Draw by using a rectangle with rounded corners.



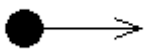
Transition:

A solid arrow represents the path between different states of an object. Label the transition with the event that triggered it and the action that results from it. Learn how to draw lines and arrows in Smart Draw.



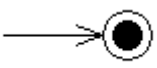
Initial State:

A filled circle followed by an arrow represents the object's initial state. Learn how to rotate objects.



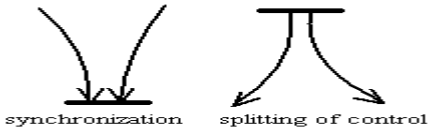
Final State:

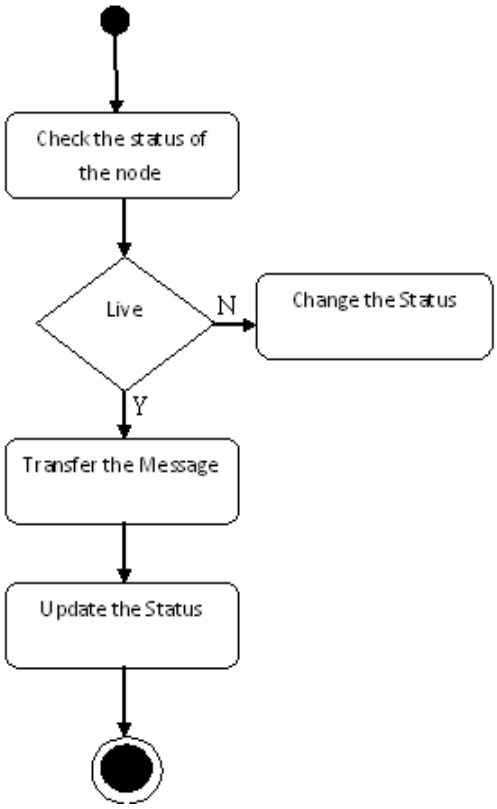
An arrow pointing to a filled circle nested inside another circle represents the object's final state.



Synchronization and Splitting of Control:

A short heavy bar with two transitions entering it represents a synchronization of control. A short heavy bar with two transitions leaving it represents a splitting of control that creates multiple states.





3.7.3.4 Activity Diagram:

Activity diagrams provide a way to model the workflow of a business process, code-specific information such as a class operation. The transitions are implicitly triggered by completion of the actions in the source activities. The main difference between activity diagrams and state charts is activity diagrams are activity centric, while state charts are state centric. An activity diagram is typically used for modeling the sequence of activities in a

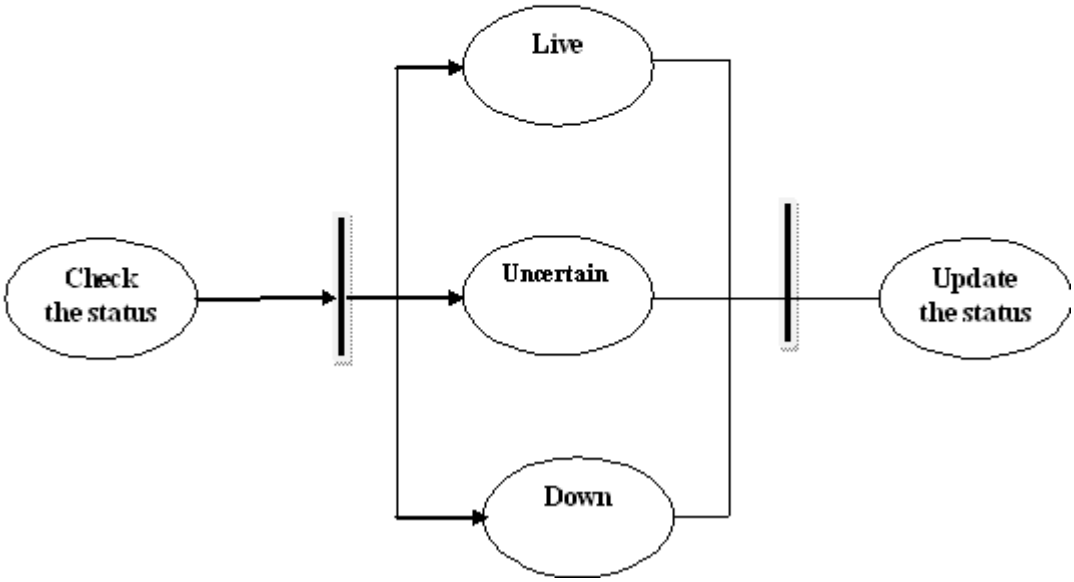
process, whereas a state chart is better suited to model the discrete stages of an object's lifetime.

An activity represents the performance of task or duty in a workflow. It may also represent the execution of a statement in a procedure. You can share activities between state machines. However, transitions cannot be shared.

An action is described as a "task" that takes place while inside a state or activity.
Actions on activities can occur at one of four times:

- **On entry**: The "task" must be performed when the object enters the state or activity.
- **On exit:** The "task" must be performed when the object exits the state or activity.
- **Do**: The "task" must be performed while in the state or activity and must continue until exiting the state.
- **On event:** The "task" triggers an action only if a specific event is received.

    - An **end state** represents a final or terminal state on an activity diagram or state chart diagram.
    - A **start state** (also called an "initial state") explicitly shows the beginning of a workflow on an activity diagram.
    - **Swim lanes** can represent organizational units or roles within a business model. They are very similar to an object. They are used to determine which unit is responsible for carrying out the specific activity. They show ownership or responsibility. Transitions cross swim lanes
    - **Synchronizations** enable you to see a simultaneous workflow in an activity diagram Synchronizations visually define forks and joins representing parallel workflow.
    - A **fork** construct is used to model a single flow of control that divides into two or more separate, but simultaneous flows. A corresponding join should ideally accompany every fork that appears on an activity diagram. A **join** consists of two of more flows of control that unite into a single flow of control. All model elements (such as activities and states) that appear between a fork and join must complete before the flow of controls can unite into one.
    - An **object flow** on an activity diagram represents the relationship between an activity and the object that creates it (as an output) or uses it (as an input).


Live
Check
the status
Uncertain
Update
the status
Down

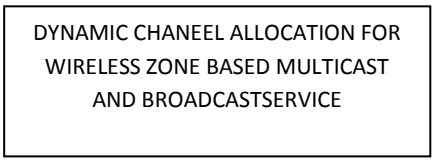
4. SYSTEM DESIGN AND DOCUMENT

4.1 System Design:

Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm and area of application. Design is the first step in the development phase for any engineered product or system. The designer's goal is to produce a model or representation of an entity that will later be built. Beginning, once system requirement have been specified and analyzed, system design is the first of the three technical activities -design, code and test that is required to build and verify software.

The importance can be stated with a single word "Quality". Design is the place where quality is fostered in software development. Design provides us with representations of software that can assess for quality. Design is the only way that we can accurately translate a customer's view into a finished software product or system. Software design serves as a foundation for all the software engineering steps that follow. Without a strong design we risk building an unstable system – one that will be difficult to test, one whose quality cannot be assessed until the last stage.

During design, progressive refinement of data structure, program structure, and procedural details are developed reviewed and documented. System design can be viewed from either technical or project management perspective. From the technical point of view, design is comprised of four activities – architectural design, data structure design, interface design and procedural design.



System design

SDLC Methodologies:

This document play a vital role in the development of life cycle (SDLC) as it describes the complete requirement of the system. It means for use by developers and will be the basic during testing phase. Any changes made to the requirements in the future will have to go through formal change approval process.

SPIRAL MODEL was defined by Barry Boehm in his 1988 article, "A spiral Model of Software Development and Enhancement. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration models.

As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with a client reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.

The steps for Spiral Model can be generalized as follows:

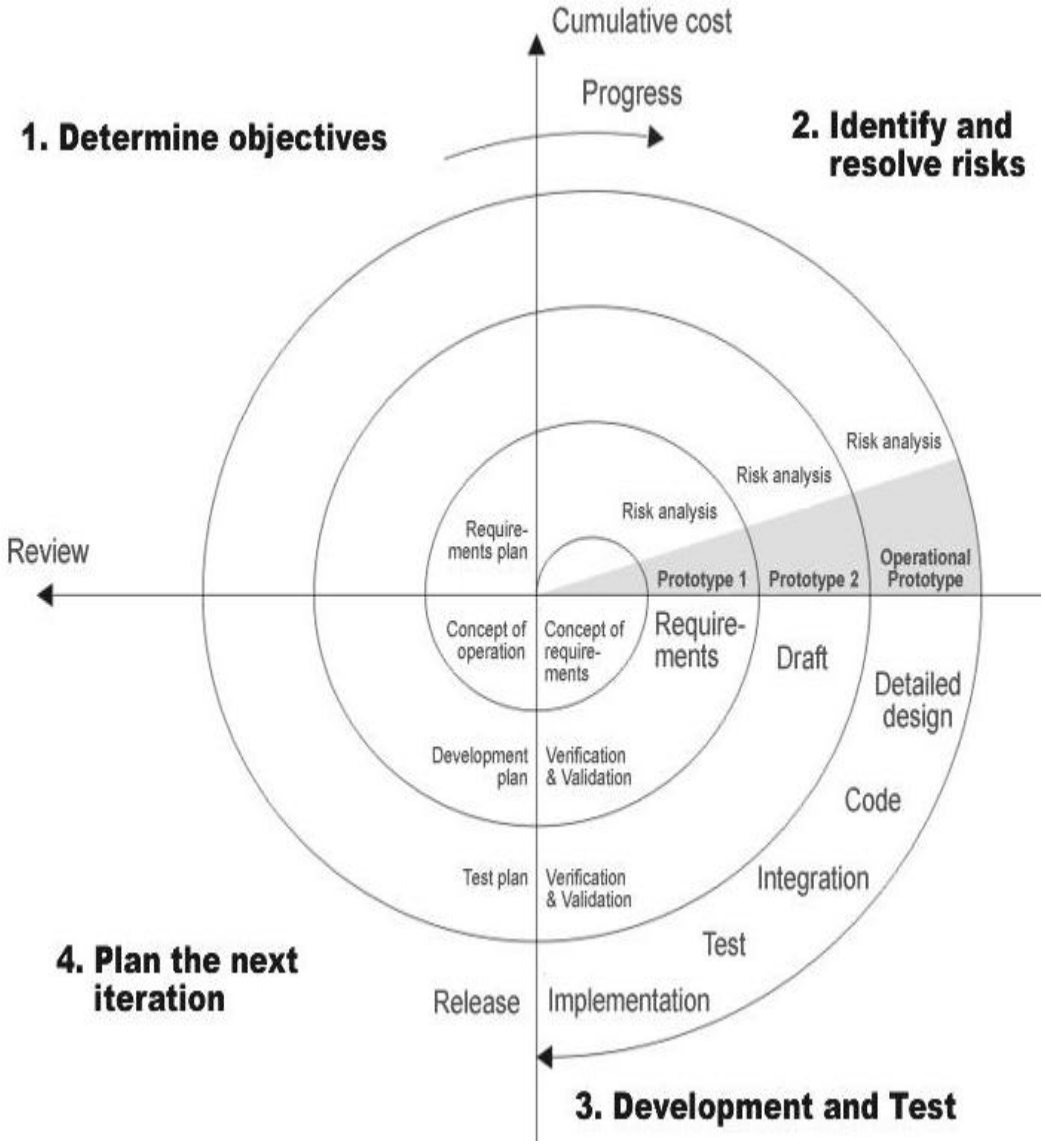
- The new system requirements are defined in as much details as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.
- A preliminary design is created for the new system.
- A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
- A second prototype is evolved by a fourfold procedure:
    1. Evaluating the first prototype in terms of its strengths, weakness, and risks.
    2. Defining the requirements of the second prototype.
    3. Planning a designing the second prototype.
    4. Constructing and testing the second prototype.
- At the customer option, the entire project can be aborted if the risk is deemed too great. Risk factors might involve development cost
- overruns, operating-cost miscalculation, or any other factor that could, in the customer's judgment, result in a less-than-satisfactory final product.
- The existing prototype is evaluated in the same manner as was the previous prototype, and if necessary, another prototype is developed from it according to the fourfold procedure outlined above.

- The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.
- The final system is constructed, based on the refined prototype.
- The final system is thoroughly evaluated and tested. Routine maintenance is carried on a continuing basis to prevent large scale failures and to minimize down time.

Advantages:

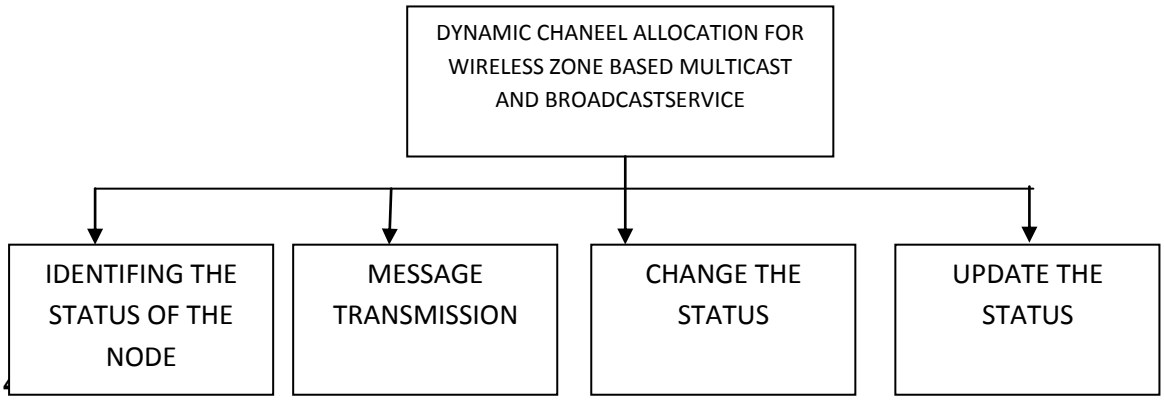
- Estimates(i.e. budget, schedule etc .) become more relistic as work progresses, because important issues discoved earlier.
- It is more able to cope with the changes that are software development generally entails.
- Software engineers can get their hands in and start woring on the core of a project earlier.

The following diagram shows how a spiral model acts like:



4.2 Sub-system Design:

When a system is large and too complex to understand we divide the core system into parts called Sub Systems



4.3 Data Flow Diagrams:

A graphical tool used to describe and analyze the moment of data through a system manual or automated including the process, stores of data, and delays in the system. Data Flow Diagrams are the central tool and the basis from which other components are developed. The transformation of data from input to output, through processes, may be described logically and independently of the physical components associated with the system. The DFD is also know as a data flow graph or a bubble chart.

Context diagram:

The top-level diagram is often called a "*context diagram"*. It contains a single process, but it plays a very important role in studying the current system. The context diagram defines the system that will be studied in the sense that it determines the boundaries. Anything that is not inside the process identified in the context diagram will not be part of the system study. It represents the entire software element as a single bubble with input and output data indicated by incoming and outgoing arrows respectively.

Types of Data Flow Diagrams:

- ⇨ **Physical DFD**
- ⇨ **Logical DFD**

1. Physical DFD

Structured analysis states that the current system should be first understand correctly. The physical DFD is the model of the current system and is used to ensure that the current system has been clearly understood. Physical DFDs shows actual devices, departments, people etc., involved in the current system

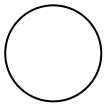
2. Logical DFD

Logical DFDs are the model of the proposed system. They clearly should show the requirements on which the new system should be built. Later during design activity this is taken as the basis for drawing the system's structure charts.

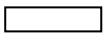
The Basic Notation used to create a DFD's are as follows:

Dataflow: Data move in a specific direction from an origin to a Destination.

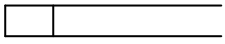
Process: People, procedures, or devices that use or produce (Transform) Data. The physical component is not identified.



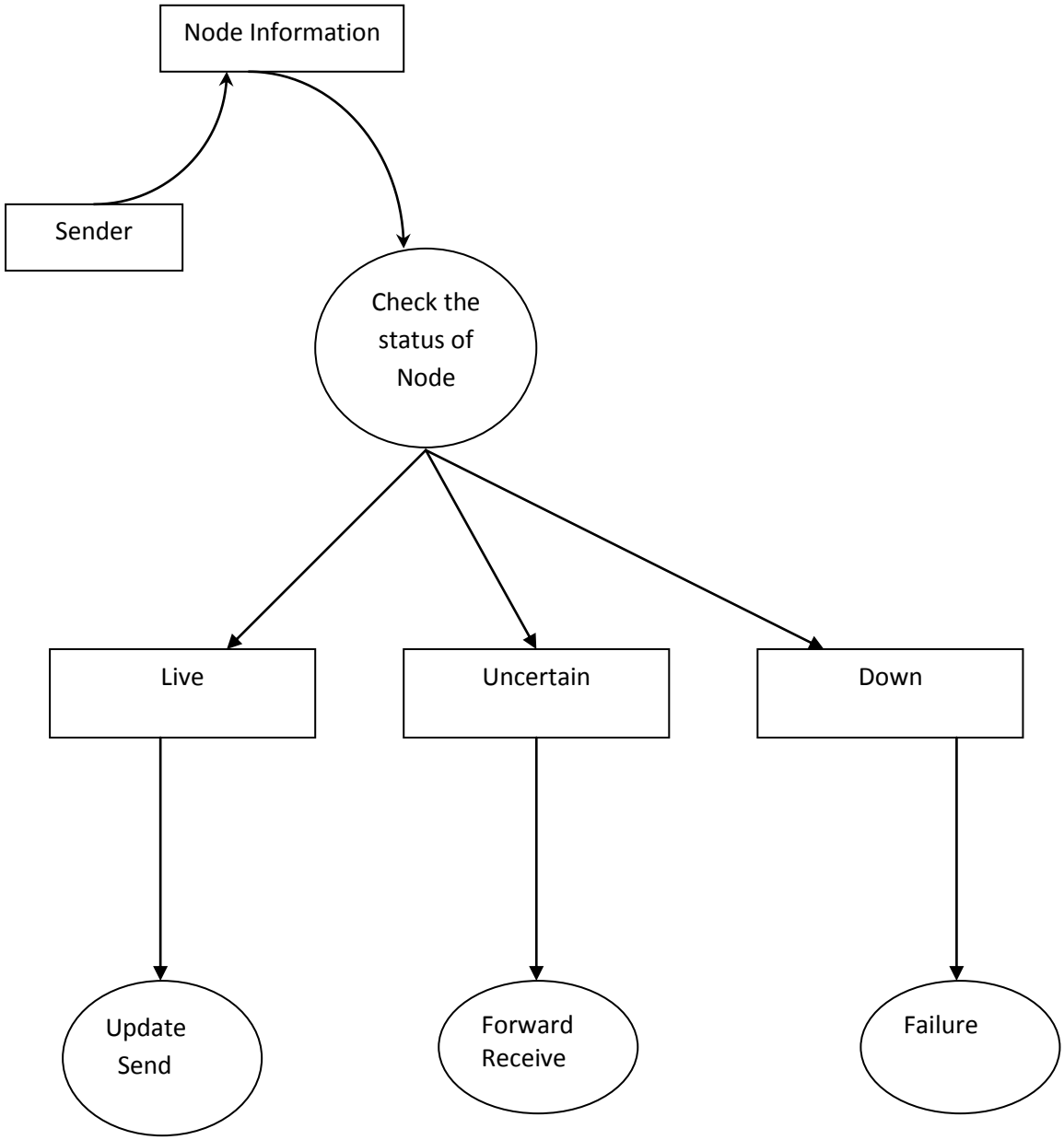
Source: External sources or destination of data, which may be People, programs, organizations or other entities.



Data Store: Here data are stored or referenced by a process in the System



Data Flow Diagrams:



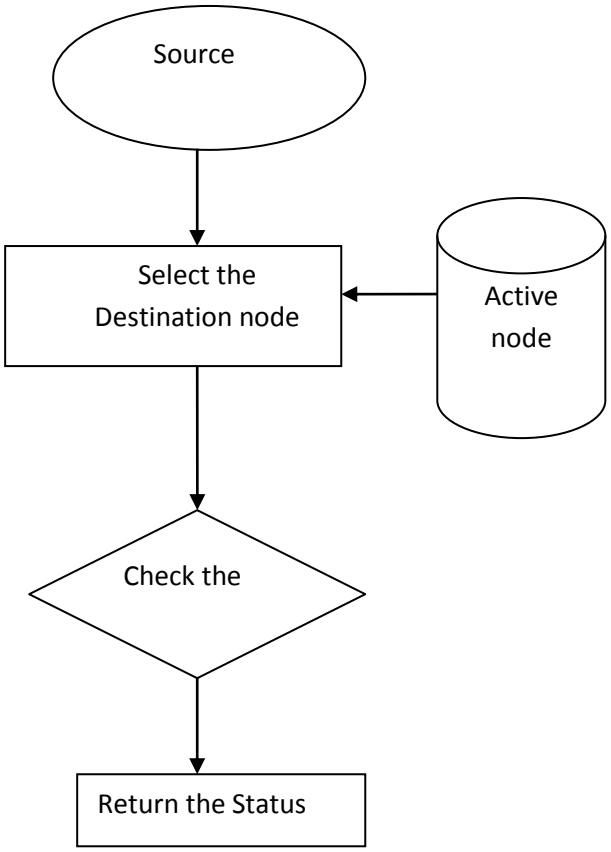


Fig: Status of the node

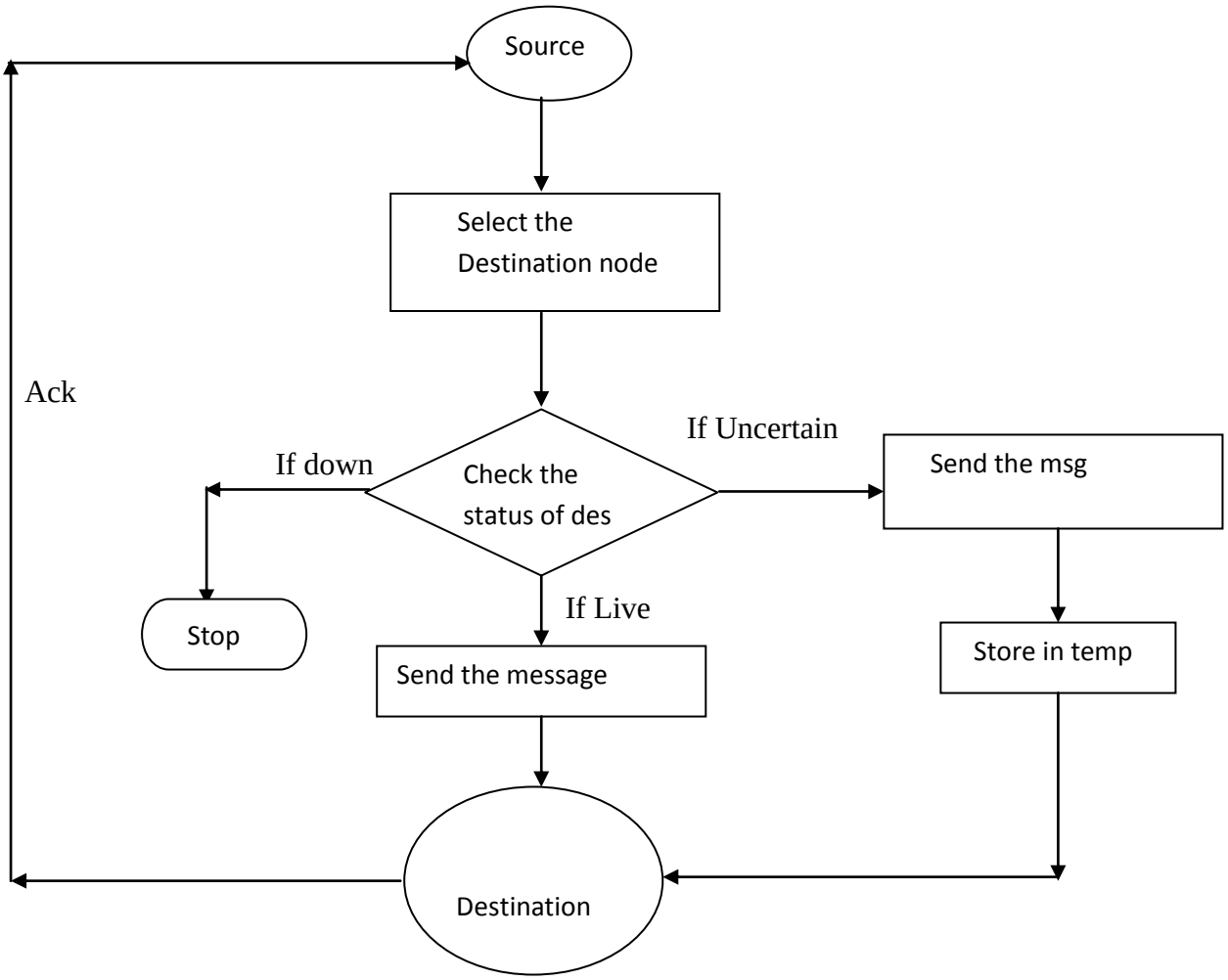


Fig Message Transmission

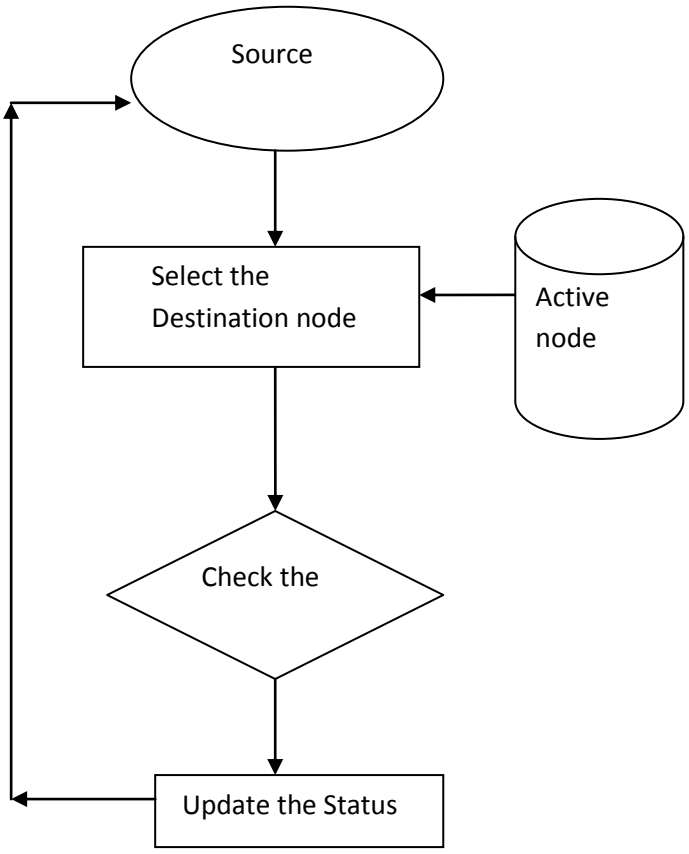


Fig Change the Status

5. SYSTEM TESTING

5.1 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 produces 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.

Functional test:

Functional tests provide 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.

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.

Performance Test

The Performance test ensures that the output be produced within the time limits, and the time taken by the system for compiling, giving response to the users and request being send to the system for to retrieve the results.

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. Integration testing for Server Synchronization: Testing the IP Address for to communicate with the other Nodes. Check the Route status in the Cache Table after the status information is received by the Node. The Messages are displayed throughout the end of the application

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.

Acceptance testing for Data Synchronization:

The Acknowledgements will be received by the Sender Node after the Packets are received by the Destination Node. The Route add operation is done only when there is a Route request in need

System 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 it's 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

5.2 Test Cases:

TEST CASE TABLE:

TABLE:

A database is a collection of data about a specific topic.

VIEWS OF TABLE:

We can work with a table in two types,

1. Design View
2. Datasheet View

Design View

To build or modify the structure of a table we work in the table design view. We can specify what kind of data will be hold.

Datasheet View

To add, edit or analyses the data itself we work in tables datasheet view mode.

QUERY:

A query is a question that has to be asked the data. Access gathers data that answers the question from one or more table. The data that make up the answer is either dynast (if you edit it) or a snapshot (it cannot be edited).Each time we run query, we get latest information in the dynast. Access either displays the dynast or snapshot for us to view or perform an action on it, such as deleting or updating.

S.No	Input	Description	Output	Result
1	Check the status	MBS zone technology will check the status.	Live ; transfer the message	Pass
2	Message transfer	It transfers The MBS content from one BS to other BS.	Changes the status of node	Pass
3	Change the status	If it is allocated for old MBS call and it task is finished and change the status of node	Update the status of the node	pass
4	Update the status	It updates the status from live to uncertain and uncertain to down and down to live	Message transferred successfully	Pass

6. SAMPLE CODING

```
//faultTolerantCN.java
import java.io.*;
import java.lang.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.Color;

public class Globals

{
        static int MobileHostID=0;
        static int MyChannelID=0;
        static int MSSWidth=10;
        static int MSSHeight=10;
        static int HexWidth=70; //cell-hexagon width
        static int HexHeight=90; //cell-hexagon height
        static int CellStartX=95; //firstcell's xpos
        static int CellStartY=140; //firstcell's ypos
class ChannelStatus

{
        static int Allocated=0;
        static int Busy=1;
        static String[] StatusString;
        {       StatusString=new String[]

                {       new String("Allocated"),
                        new String("Free"),
                };
        }
}
class MobileHost
{
        int id;
        int x;
        int y;
        MobileHost(int tx,int ty)
        {
                id=Globals.MobileHostID;
                x=tx;
                y=ty;
                Globals.MobileHostID++;
```

```
        }
        public int getid()
        {       return(id);
        }
        public int getx()
        {       return(x);
        }

        public int gety()
        {       return(y);
        }
}
class MyChannel
{
        int id;
        int x;
        int y;
        int status;
        MyChannel(int tx,int ty,int tstatus)
        {
                id=Globals.MyChannelID;
                Globals.MyChannelID++;
                x=tx;
                y=ty;
                status=tstatus;
        }
        public int getx()
        {       return(x);
        }
        public int gety()
        {       return(y);
        }
        public int getstatus()
        {       return(status);
        }
}
class Cell
{
        int id;             //unique id for each cell
        int x,y;  //display position
        int AdjCount;
        int AdjCells[]=new int[6]; //maximum 6 adjacent cells for a cell
        MobileHost[] MobileHosts=new MobileHost[100];
        int MobileHostCount;
        MyChannel[] MyChannels=new MyChannel[100];
        int MyChannelCount;
        int LinkedMH[]=new int[100];
```

```
int GroupCells[]=new int[100];
int GroupCellCount;
//constructor
Cell()
{
        id=-1;
        AdjCount=0;
        MobileHostCount=0;
        MyChannelCount=0;
        GroupCellCount=0;
}

//set functions
public void setid(int tid)
{
        id=tid;
}
public void setxy(int tx,int ty)
{       x=tx;
        y=ty;
}
//get functions
public int getid()
{       return(id);
}
public int getx()
{       return(x);
}
public int gety()
{       return(y);
}
public int getMobileHostCount() //MobileHosts get functions
{       return(MobileHostCount);
}

public MobileHost getMobileHost(int index)
{       return(MobileHosts[index]);
}
public int getMobileHostID(int index)
{       return(MobileHosts[index].id);
}
public int getMobileHostX(int index)
{       return(MobileHosts[index].x);
}

public int getMobileHostY(int index)
{       return(MobileHosts[index].y);
```

```
}
public int getMyChannelCount() //MyChannels get functions
{       return(MyChannelCount);
}
public MyChannel getMyChannel(int index)
{       return(MyChannels[index]);
}
public int getMyChannelLink(int index)
{       return(LinkedMH[index]);
}
public int getMyChannelID(int index)
{       return(MyChannels[index].id);
}
public int getMyChannelX(int index)
{       return(MyChannels[index].x);
}
public int getMyChannelY(int index)
{       return(MyChannels[index].y);
}
public int getMyChannelStatus(int index)
{       return(MyChannels[index].status);
}
public void setMyChannelStatus(int tIndex,int tStatus)
{       MyChannels[tIndex].status=tStatus;
}

public int getAdjCount() //AdjCell get functions
{       return(AdjCount);
}
public int getAdjCell(int tindex)
{       int tcellid=-1;
        if(tindex>=0&&tindex<=AdjCount-1) tcellid=AdjCells[tindex];
        return(tcellid);
}
public int getGroupCell(int tindex)
{       int tcellid=-1;
        if(tindex>=0&&tindex<=GroupCellCount-1) tcellid=GroupCells[tindex];
        return(tcellid);
}
//methods
public void addAdjCell(int tcellid)
{       AdjCells[AdjCount]=tcellid;
        AdjCount++;
}
public void addMobileHost(int xposition,int yposition)
{       MobileHosts[MobileHostCount]=new MobileHost(xposition,yposition);
        LinkedMH[MobileHostCount]=-1;
```

```
		MobileHostCount++;
	}
	public boolean addMyChannel(int tx,int ty,int tstatus)
	{	boolean flag=false;
		if(MobileHostCount!=0)
		{		if(MyChannelCount<MobileHostCount)
			{
				MyChannels[MyChannelCount]=new
MyChannel(tx,ty,tstatus);

	LinkedMH[MyChannelCount]=MobileHosts[MyChannelCount].getid();
				MyChannelCount++;
				flag=true;
			}
		}
		//return(flag);
		return(true);
	}
	public void addGroupCell(int tCellID)
	{	GroupCells[GroupCellCount]=tCellID;
		GroupCellCount++;
	}
	public void draw(Graphics g)
	{	//draw cell-hexagon
		g.setColor(new Color(0,0,255));
		Graphics2D g2d=(Graphics2D)g;
		g2d.setStroke(new BasicStroke(2.0f));
		g.drawLine(x,y,x+Globals.HexWidth,y);
	g.drawLine(x+Globals.HexWidth,y,x+Globals.HexWidth+Globals.HexWidth/2,y+Gl
obals.HexHeight/2);
		g.drawLine(x,y,x-Globals.HexWidth/2,y+Globals.HexHeight/2);
		g.drawLine(x-
Globals.HexWidth/2,y+Globals.HexHeight/2,x,y+Globals.HexHeight);
	g.drawLine(x+Globals.HexWidth+Globals.HexWidth/2,y+Globals.HexHeight/2,x+G
lobals.HexWidth,y+Globals.HexHeight);
	g.drawLine(x,y+Globals.HexHeight,x+Globals.HexWidth,y+Globals.HexHeight);

		//draw MSS
		int tCenterX=(Globals.HexWidth-Globals.MSSWidth)/2;
		int tCenterY=(Globals.HexHeight-Globals.MSSHeight)/2;

	g.drawRect(x+tCenterX,y+tCenterY,Globals.MSSWidth,Globals.MSSHeight);
		try
		{	//Thread.sleep(1);
		}
		catch(Exception e)
		{	System.out.println("Error: "+e.getMessage());
```

```
                }
        }
};
class Map
{       int CellCount=7;        //number of cells
        Cell MapCells[];        //cells list
        //constructor
        Map()
        {       MapCells=new Cell[CellCount];
                //set cell ids
                for(int t=0;t<CellCount;t++)
                {       MapCells[t]=new Cell();
                        MapCells[t].setid(t);
                }
                //set adjcent cells
                MapCells[0].addAdjCell(1);      //
                MapCells[0].addAdjCell(4);
                MapCells[0].addAdjCell(3);
                MapCells[1].addAdjCell(0);      //
                MapCells[1].addAdjCell(4);
                MapCells[1].addAdjCell(2);
                MapCells[2].addAdjCell(1);      //
                MapCells[2].addAdjCell(4);
                MapCells[2].addAdjCell(5);
                MapCells[3].addAdjCell(0);      //
                MapCells[3].addAdjCell(4);
                MapCells[3].addAdjCell(6);
                MapCells[4].addAdjCell(0);      //
                MapCells[4].addAdjCell(1);
                MapCells[4].addAdjCell(2);
                MapCells[4].addAdjCell(3);
                MapCells[4].addAdjCell(6);
                MapCells[4].addAdjCell(5);
                MapCells[5].addAdjCell(2);      //
                MapCells[5].addAdjCell(4);
                MapCells[5].addAdjCell(6);
                MapCells[6].addAdjCell(3);      //
                MapCells[6].addAdjCell(4);
                MapCells[6].addAdjCell(5);
                //set cells screen pos
                MapCells[0].setxy(Globals.CellStartX,Globals.CellStartY);
        MapCells[1].setxy(Globals.CellStartX+Globals.HexWidth+Globals.HexWidth/2,Glo
bals.CellStartY-Globals.HexHeight/2);
        MapCells[2].setxy(Globals.CellStartX+Globals.HexWidth*3,Globals.CellStartY);

        MapCells[3].setxy(Globals.CellStartX,Globals.CellStartY+Globals.HexHeight);
```

```
        MapCells[4].setxy(Globals.CellStartX+Globals.HexWidth+Globals.HexWidth/2,Glo
bals.CellStartY+Globals.HexHeight/2);
        MapCells[5].setxy(Globals.CellStartX+Globals.HexWidth*3,Globals.CellStartY+Glo
bals.HexHeight);
        MapCells[6].setxy(Globals.CellStartX+Globals.HexWidth+Globals.HexWidth/2,Glo
bals.CellStartY+Globals.HexHeight+Globals.HexHeight/2);
                //set cell groups
                MapCells[0].addGroupCell(2); //
                MapCells[0].addGroupCell(5);
                MapCells[0].addGroupCell(6);
                MapCells[1].addGroupCell(3); //
                MapCells[1].addGroupCell(5);
                MapCells[1].addGroupCell(6);
                MapCells[2].addGroupCell(0); //
                MapCells[2].addGroupCell(3);
                MapCells[2].addGroupCell(6);
                MapCells[3].addGroupCell(1); //
                MapCells[3].addGroupCell(2);
                MapCells[3].addGroupCell(5);
                MapCells[5].addGroupCell(0); //
                MapCells[5].addGroupCell(1);
                MapCells[5].addGroupCell(3);
                MapCells[6].addGroupCell(0); //
                MapCells[6].addGroupCell(1);
                MapCells[6].addGroupCell(2);
        }
        //get functions
        public Cell[] getMapCells()
        {       return(MapCells); }
        public int getCellCount()
        {       return(CellCount);
        }
        public int getTotalMobileHostsCount()
        {       int tCount=0;
                for(int x=0;x<CellCount;x++)
                {       tCount+=MapCells[x].getMobileHostCount();
                }
                return(tCount);  }
        public int getTotalMyChannelsCount()
        {       int tCount=0;
                for(int x=0;x<CellCount;x++)
                {       tCount+=MapCells[x].getMyChannelCount();
                }
                return(tCount);
        }
        //returns which cell a mobileHost is present
        public int getCellIDOfMobileHost(int tMobileHostID)
        {       int tCellID=-1;
                for(int t=0;t<CellCount;t++)
                {       for(int j=0;j<MapCells[t].getMobileHostCount();j++)
                        {       if(MapCells[t].getMobileHostID(j)==tMobileHostID)
                                {       tCellID=t;
                                }
                        }
```

```
		}
		return(tCellID);
	}
	//returns index of a mobileHost inside a cell
	public int getMobileHostIndexInCell(int tMobileHostID)
	{	int tIndex=-1;
		for(int t=0;t<CellCount;t++)
		{	for(int j=0;j<MapCells[t].getMobileHostCount();j++)
			{	if(MapCells[t].getMobileHostID(j)==tMobileHostID)
				{	tIndex=j;
				}
			}
		}
		return(tIndex);
	}
	//returns which cell a mychannel is present
	public int getCellIDOfMyChannel(int tMyChannelID)
	{	int tCellID=-1;
		for(int t=0;t<CellCount;t++)
		{	for(int j=0;j<MapCells[t].getMyChannelCount();j++)
			{	if(MapCells[t].getMyChannelID(j)==tMyChannelID)
				{	tCellID=t;
				}
			}
		}
		return(tCellID);
	}
	//returns index of a mychannel inside a cell
	public int getMyChannelIndexInCell(int tMyChannelID)
	{	int tIndex=-1;
		for(int t=0;t<CellCount;t++)
		{	for(int j=0;j<MapCells[t].getMyChannelCount();j++)
			{	if(MapCells[t].getMyChannelID(j)==tMyChannelID)
				{	tIndex=j;
				}
			}
		}
		return(tIndex);
	}
	//methods
	public void draw(Graphics g)
	{	try
		{	//Thread.sleep(1);
			for(int t=0;t<CellCount;t++) MapCells[t].draw(g);
		}catch(Exception e)
		{	System.out.println("Error: "+e.getMessage());	}	}
	//draw mobileHosts
	public void drawMobileHostsLabel(Graphics g)
	{	g.setFont(new Font("Verdana",Font.BOLD,13));
		g.setColor(Color.BLUE);
		for(int i=0;i<CellCount;i++)
		{	g.drawString("cell"+i,MapCells[i].getx()+20,MapCells[i].gety()+12);
		}
	}
```

```
	public String getAdjListString(int tCellIndex)
	{	String tstr="";
		if(MapCells[tCellIndex].getAdjCount()!=0)
		{	for(int ti=0;ti<MapCells[tCellIndex].getAdjCount();ti++)
			{
	tstr=tstr+Integer.toString(MapCells[tCellIndex].getAdjCell(ti))+" ";
			}
		}			return(tstr);
	}



public String getCellInfo()

	{	//adjlist
		String tstr="";
		for(int t=0;t<CellCount;t++)
		{	tstr+="cell"+MapCells[t].getid()+": [";
			tstr+=getAdjListString(t)+"]; ";
			tstr+=""+MapCells[t].getMobileHostCount()+" MHs; ";
			tstr+=""+MapCells[t].getMyChannelCount()+" Channels.";
			tstr+="\n";
		}			return(tstr);
	}
};

class faultTolerantCN extends JFrame implements
ActionListener,MouseMotionListener,MouseListener
{	//ui
	JFrame frmMain=new JFrame("Dynamic Channel Allocation for Wireless Zone-
Based Multicast and Broadcast Service");
	JFrame frmResult=new JFrame("Result");
	JTextArea txtCellInfo=new JTextArea("");
	JTextArea txtChannelInfo=new JTextArea("");
	JTextArea txtDataStructures=new JTextArea("");
	JTextArea txtResult=new JTextArea("");
	JScrollPane spCellInfo=new JScrollPane(txtCellInfo);
	JScrollPane spChannelInfo=new JScrollPane(txtChannelInfo);
	JScrollPane spDataStructures=new JScrollPane(txtDataStructures);
	JScrollPane spResult=new JScrollPane(txtResult);
	JButton btStart=new JButton("Start");
	JLabel lblSimulationTime=new JLabel("");
	JLabel lblMobileHostCount=new JLabel("");
	JLabel lblXY=new JLabel("");
	JLabel lblCurrentMobileHost=new JLabel("");
	JLabel lblSourceMobileHost=new JLabel("");
	JLabel lblCurrentMyChannel=new JLabel("");
	JLabel lblAllocated=new JLabel("Allocated-Channel");
	JLabel lblBusy=new JLabel("Free-Channel");
	JLabel lblRequestToMSS=new JLabel("Request-to-MSS");
	JLabel lblRequestToNeighbor=new JLabel("Request-to-Neighbor");
	Graphics g;
	//ui parameters
```

```
        int frmWidth=1200,frmHeight=480;
        int mapWidth=420,mapHeight=340;
        //system parameters
        int currentMobileHost=-1;
        int sourceMobileHost=-1;
        int currentMyChannel=-1;
        boolean sourceSelected=false;
        int maxMobileHosts;
        int maxMyChannels;
        int radius=5;
        boolean ChannelAllocated=false;
        //channel allocation parameters
        boolean RequestMessageSent=false;
        Map map=new Map();
        Cell[] MapCells=map.getMapCells();
        javax.swing.Timer timerMain;
        long maptime=0;
        //constructor
        faultTolerantCN()
        {       frmMain.getContentPane().setBackground(new Color(153,153,51));
                //get user-paramters
                maxMobileHosts=Integer.parseInt(JOptionPane.showInputDialog("Enter
MobileHostCount: ","20"));
                maxMyChannels=Integer.parseInt(JOptionPane.showInputDialog("Enter
ChannelCount: ","10"));
                frmMain.setDefaultLookAndFeelDecorated(true);
                //frmMain.setResizable(false);
                frmMain.setBounds(20,60,frmWidth,frmHeight);
                frmMain.getContentPane().setLayout(null);
                frmMain.addMouseMotionListener(this);
                frmMain.addMouseListener(this);
                frmResult.setDefaultLookAndFeelDecorated(true);
                frmResult.setResizable(false);
                frmResult.getContentPane().setBackground(new Color(153,153,51));
                frmResult.setBounds(20+frmWidth+10,60,240,frmHeight);
                frmResult.getContentPane().setLayout(null);

                txtCellInfo.setEditable(false);
                spCellInfo.setBounds(frmWidth-450-250,35,200,300);
                frmMain.getContentPane().add(spCellInfo);
                txtDataStructures.setEditable(false);
                spDataStructures.setBounds(frmWidth-450-30,35,200,300);
                frmMain.getContentPane().add(spDataStructures);
                txtChannelInfo.setEditable(false);
                spChannelInfo.setBounds(frmWidth-230-30,35,200,140);
                frmMain.getContentPane().add(spChannelInfo);
                txtResult.setEditable(false);
                spResult.setBounds(20,20,200,frmHeight-160);
                frmResult.getContentPane().add(spResult);
                txtResult.setText("Result:\n");
                btStart.setBounds(frmWidth-240-30,280,200,25);
                frmMain.getContentPane().add(btStart);
                btStart.addActionListener(this);
                lblSimulationTime.setBounds(950,frmHeight-300,150,20);
```

```
frmMain.getContentPane().add(lblSimulationTime);
lblMobileHostCount.setBounds(950,frmHeight-280,150,20);
frmMain.getContentPane().add(lblMobileHostCount);
lblXY.setBounds(30,10,100,20);
frmMain.getContentPane().add(lblXY);
lblSourceMobileHost.setBounds(frmWidth-430,10,180,20);
frmMain.getContentPane().add(lblSourceMobileHost);
lblCurrentMobileHost.setBounds(frmWidth-250,frmHeight-260,180,20);
frmMain.getContentPane().add(lblCurrentMobileHost);
lblCurrentMyChannel.setBounds(frmWidth-250,frmHeight-240,200,20);
frmMain.getContentPane().add(lblCurrentMyChannel);



        //set indicator labels
lblAllocated.setBounds(70,frmHeight-130,150,20);
frmResult.getContentPane().add(lblAllocated);
lblBusy.setBounds(70,frmHeight-110,150,20);
frmResult.getContentPane().add(lblBusy);
lblRequestToMSS.setBounds(70,frmHeight-90,150,20);
frmResult.getContentPane().add(lblRequestToMSS);
lblRequestToNeighbor.setBounds(70,frmHeight-70,150,20);
frmResult.getContentPane().add(lblRequestToNeighbor);
frmMain.setVisible(true);
frmResult.setVisible(true);
try
{       //Thread.sleep(1);
} catch(Exception e) { }
g=frmMain.getGraphics();
Graphics2D g2d=(Graphics2D)g;
g2d.setStroke(new BasicStroke(2.0f));
g2d.drawRect(30,60,mapWidth,mapHeight);
Image i=new ImageIcon("images\\cbe_district_map.gif").getImage();
//Image i=new ImageIcon("images\\cbe_district_map.gif").getImage();
//Image i=new ImageIcon("images\\cbe_city_map.jpg").getImage();
g.drawImage(i,30,60,mapWidth,mapHeight,this);
map.draw(g);
map.drawMobileHostsLabel(g);
        //set Indicators
Graphics g1=frmResult.getGraphics();
g1.setColor(new Color(0,0,0)); //allocated
g1.drawRect(20,frmHeight-120+30,40,2);
g1.setColor(new Color(255,0,255)); //busy
g1.drawRect(20,frmHeight-100+30,40,2);
g1.setColor(new Color(0,255,0)); //request-to-mss
g1.drawRect(20,frmHeight-80+30,40,2);
g1.setColor(new Color(0,0,255)); //request-to-neighbor
g1.drawRect(20,frmHeight-60+30,40,2);
//set timer
timerMain=new javax.swing.Timer(1000,timerMain_Tick);
timerMain.start();
while(true);
}
```

```
	public Map getMap()
	{	return(map);
	}
	//events
	ActionListener timerMain_Tick=new ActionListener()
	{	public void actionPerformed(ActionEvent e)
		{	//add mobilehosts
			if(map.getTotalMobileHostsCount()>=maxMobileHosts)
			{	timerMain.stop();
				addRandomMyChannels(maxMyChannels);
				displayStatus();
				updateChannelInfo();
				updateDataStructures();
				drawChannelLinks(4);
			}	else
			{	maptime++;
		lblSimulationTime.setText("Simulation Time: "+Long.toString(maptime));
				addRandomMobileHosts(5);
				lblMobileHostCount.setText("Total MobileHosts:
"+map.getTotalMobileHostsCount());
			}
		}
	};
	public void actionPerformed(ActionEvent evt)
	{	if(evt.getSource()==btStart)
		{	if(sourceMobileHost!=-1)
			{	SendRequestMessage();
			int  tLocalCellID=map.getCellIDOfMobileHost(sourceMobileHost);
				CheckChannelsOfCell(tLocalCellID);
				if(ChannelAllocated==false)
					{	addResultText("\n\nSending Request To
		Neighbors of "+tLocalCellID+":
		["+map.getAdjListString(tLocalCellID)+"]");
		if(MapCells[tLocalCellID].getAdjCount()>0)
					{	for(int
t=0;t<MapCells[tLocalCellID].getAdjCount();t++)
					{

	DrawCellRequestArrow(tLocalCellID,MapCells[tLocalCellID].getAdjCell(t));

	CheckChannelsOfCell(MapCells[tLocalCellID].getAdjCell(t));
							if(ChannelAllocated==true) break;
					}
				}
			}
		}
	}
	void SendRequestMessage()
	{	if(RequestMessageSent==false)
		{
			addResultText("\nMH"+sourceMobileHost+" Sending Request
Message to MSS"+map.getCellIDOfMobileHost(sourceMobileHost));
```

```
                        int
x1=MapCells[map.getCellIDOfMobileHost(sourceMobileHost)].getMobileHostX(map.getM
obileHostIndexInCell(sourceMobileHost));
                        int
y1=MapCells[map.getCellIDOfMobileHost(sourceMobileHost)].getMobileHostY(map.getM
obileHostIndexInCell(sourceMobileHost));
                        int
x2=MapCells[map.getCellIDOfMobileHost(sourceMobileHost)].getx();
                        int
y2=MapCells[map.getCellIDOfMobileHost(sourceMobileHost)].gety();
                        g.setColor(Color.GREEN);
                        int tCenterX=Globals.HexWidth/2;
                        int tCenterY=Globals.HexHeight/2;
                        g.drawLine(x1,y1,x2+tCenterX,y2+tCenterY);
                        RequestMessageSent=true;
                }
        }
        void CheckChannelsOfCell(int tCellID)
        {
                addResultText("\n\nChecking Channels
["+getChannelsOfCell_String(tCellID)+"] of Cell"+tCellID);
                int tCount=MapCells[tCellID].getMyChannelCount();
                int selectedChannel=-1;
                if(tCount!=0)
                {       for(int t=0;t<tCount;t++)
                        {
                                int tChannelID=MapCells[tCellID].getMyChannelID(t);
                                //DrawChannelRequestArrow(tCellID,tChannelID);

        if(MapCells[tCellID].getMyChannelStatus(t)==ChannelStatus.Allocated)
                                {
                                        if(selectedChannel==-1)
selectedChannel=tChannelID;
                                }
                        }
                }
                if(selectedChannel==-1)
                {
                        addResultText("\nNo Allocated Channels in Cell"+tCellID);
                }
                else
                {
                        ChannelAllocated=true;
                        ChangeChannelStatus(selectedChannel,ChannelStatus.Busy);
                        addResultText("\nChannel"+selectedChannel+" allocated");
                        updateChannelInfo();
                        updateDataStructures();
                        HighlightAllocatedChannel(selectedChannel);
                }
        }

        void ChangeChannelStatus(int tChannelID,int tNewStatus)
        {
```

```
        MapCells[map.getCellIDOfMyChannel(tChannelID)].setMyChannelStatus(map.get
MyChannelIndexInCell(tChannelID),tNewStatus);
                updateChannelInfo();
        }
                void DrawChannelRequestArrow(int tCellID,int tChannelID)
        {
                int tCenterX=Globals.HexWidth/2;
                int tCenterY=Globals.HexHeight/2;
                int x1=MapCells[tCellID].getx()+tCenterX;
                int y1=MapCells[tCellID].gety()+tCenterY;
                int
x2=MapCells[map.getCellIDOfMyChannel(tChannelID)].getMyChannelX(map.getMyChann
elIndexInCell(tChannelID));
                int
y2=MapCells[map.getCellIDOfMyChannel(tChannelID)].getMyChannelY(map.getMyChann
elIndexInCell(tChannelID));
                g.setColor(new Color(128,0,0));
                g.drawLine(x1,y1,x2,y2);
        }

        void DrawCellRequestArrow(int tCellID1,int tCellID2)
        {
                int tCenterX=Globals.HexWidth/2;
                int tCenterY=Globals.HexHeight/2;
                int x1=MapCells[tCellID1].getx()+tCenterX;
                int y1=MapCells[tCellID1].gety()+tCenterY;
                int x2=MapCells[tCellID2].getx()+tCenterX;
                int y2=MapCells[tCellID2].gety()+tCenterY;
                g.setColor(new Color(0,0,255));
                g.drawLine(x1,y1,x2,y2);
        }

        void HighlightAllocatedChannel(int tChannelID)
        {
                int
tx=MapCells[map.getCellIDOfMyChannel(tChannelID)].getMyChannelX(map.getMyChann
elIndexInCell(tChannelID));
                int
ty=MapCells[map.getCellIDOfMyChannel(tChannelID)].getMyChannelY(map.getMyChann
elIndexInCell(tChannelID));
                g.setColor(new Color(255,0,255));
                g.drawOval(tx-4,ty-4,10,10);
        }
        ArrayList getChannelsOfCell(int tCellID)
        {       ArrayList alChannels=new ArrayList();
                        for(int t=0;t<MapCells[tCellID].getMyChannelCount();t++)
                {
                        int tChannelID=MapCells[tCellID].getMyChannelID(t);
                        Integer int1=new Integer(tChannelID);
                        alChannels.add(int1);
                }       return(alChannels);
        }
        String getChannelsOfCell_String(int tCellID)
```

```
	{	String tstr="";
		ArrayList al1=getChannelsOfCell(tCellID);
		for(int t=0;t<al1.size();t++) tstr+=((Integer)al1.get(t)).intValue()+",";
		if(tstr.length()!=0) tstr=tstr.substring(0,tstr.length()-1);
		return(tstr);
	}
	ArrayList getChannelsInStatus(int tStatus)
	{	ArrayList alStatus=new ArrayList();
			for(int t=0;t<map.getTotalMyChannelsCount();t++)
		{				int
tStatus1=MapCells[map.getCellIDOfMyChannel(t)].getMyChannelStatus(map.getMyChanne
lIndexInCell(t));
			if(tStatus==tStatus1)
			{	Integer int1=new Integer(t);
				alStatus.add(int1);
			}
		}	return(alStatus);
	}
		String getChannelsInStatus_String(int tStatus)
	{
		String tstr="";
		ArrayList al1=getChannelsInStatus(tStatus);
		for(int t=0;t<al1.size();t++) tstr+=((Integer)al1.get(t)).intValue()+",";
		if(tstr.length()!=0) tstr=tstr.substring(0,tstr.length()-1);
		return(tstr);
	}
	ArrayList getChannelsOfCellInStatus(int tCellID,int tStatus)
	{	ArrayList alChannels=new ArrayList();
			for(int t=0;t<MapCells[tCellID].getMyChannelCount();t++)
		{	int tStatus1=MapCells[tCellID].getMyChannelStatus(t);
			if(tStatus1==tStatus)
			{
				int tChannelID=MapCells[tCellID].getMyChannelID(t);
				Integer int1=new Integer(tChannelID);
				alChannels.add(int1);
			}
		}	return(alChannels);
	}
	String getChannelsOfCellInStatus_String(int tCellID,int tStatus)
	{
		String tstr="";
		ArrayList al1=getChannelsOfCellInStatus(tCellID,tStatus);
		for(int t=0;t<al1.size();t++) tstr+=((Integer)al1.get(t)).intValue()+",";
		if(tstr.length()!=0) tstr=tstr.substring(0,tstr.length()-1);
		return(tstr);
	}
	void updateChannelInfo()
	{	String tstr="Channel Info:\n\n";
		tstr+="Allocated:
"+getChannelsInStatus_String(ChannelStatus.Allocated)+"\n";
		tstr+="Free: "+getChannelsInStatus_String(ChannelStatus.Busy)+"\n";
		txtChannelInfo.setText(tstr);
		txtCellInfo.setText("Cell-Info:\n\n"+map.getCellInfo());
	}
```

```
	void updateDataStructures()
	{		String tstr="";
			for(int t=0;t<map.getCellCount();t++)
		{		tstr+="Cell"+t+":\n";
			tstr+="Allocated("+t+"):
"+getChannelsOfCellInStatus_String(t,ChannelStatus.Allocated)+"\n";
			tstr+="Free("+t+"):
"+getChannelsOfCellInStatus_String(t,ChannelStatus.Busy)+"\n\n";
		}

			txtDataStructures.setText("Data Structures:\n\n"+tstr);
	}
			void drawChannelLinks(int tCellID)
	{		for(int j=0;j<MapCells[tCellID].getMyChannelCount();j++)
			{	int tStatus1=MapCells[tCellID].getMyChannelStatus(j);
				if(tStatus1==ChannelStatus.Busy)
				{	g.setColor(new Color(255,0,255));
				}
				else if(tStatus1==ChannelStatus.Allocated)
				{	g.setColor(new Color(0,0,0));
				}
			int tMobileHost=MapCells[tCellID].getMyChannelLink(j);
				int
x1=MapCells[tCellID].getMobileHostX(map.getMobileHostIndexInCell(tMobileHost));
				int
y1=MapCells[tCellID].getMobileHostY(map.getMobileHostIndexInCell(tMobileHost));
				int x2=MapCells[tCellID].getMyChannelX(j);
				int y2=MapCells[tCellID].getMyChannelY(j);
				g.drawLine(x1,y1,x2,y2);
			}
	}
	void swapChannelStatus(int tChannelID)
	{	int tCellID=map.getCellIDOfMyChannel(tChannelID);
			int
tStatus=MapCells[tCellID].getMyChannelStatus(map.getMyChannelIndexInCell(tChannelID
));
	ChangeChannelStatus(tChannelID,tStatus==ChannelStatus.Allocated?ChannelStatus.
Busy:ChannelStatus.Allocated);
			updateChannelInfo();
			updateDataStructures();
			drawChannelLinks(tCellID);
			displayStatus();
	}

//algorithms
	void drawPath(int tindex1,int tindex2)
	{
			int
x1=MapCells[map.getCellIDOfMobileHost(tindex1)].getMobileHostX(map.getMobileHostIn
dexInCell(tindex1));
			int
y1=MapCells[map.getCellIDOfMobileHost(tindex1)].getMobileHostY(map.getMobileHostIn
dexInCell(tindex1));
```

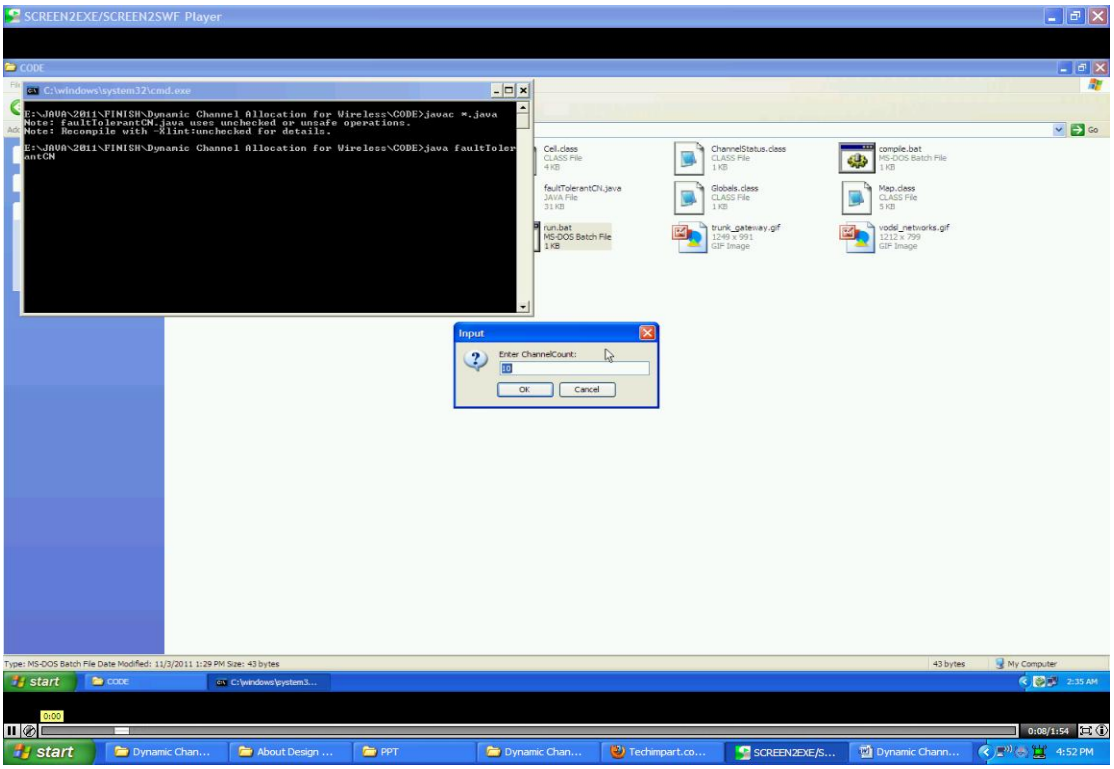
```
                int
x2=MapCells[map.getCellIDOfMobileHost(tindex2)].getMobileHostX(map.getMobileHostIn
dexInCell(tindex2));
                int
y2=MapCells[map.getCellIDOfMobileHost(tindex2)].getMobileHostY(map.getMobileHostIn
dexInCell(tindex2));
                g.setColor(Color.GREEN);
                g.drawLine(x1,y1,x2,y2);
        }
        //find euclidean distance between geographical mobilehosts
        int getDistance(int x1,int y1,int x2,int y2)
        {       double d=Math.sqrt(Math.pow((double)(x2-
x1),2.0)+Math.pow((double)(y2-y1),2.0));
                return((int)d);
        }
        //utility functions
        boolean between(int x,int y,int x1,int y1,int x2,int y2)
        {       boolean flag=false;
                if(x>=x1&&x<=x2)
                {       if(y>=y1&&y<=y2)
                        {       flag=true;
                        }
                }       return(flag);
        }


        public void displayStatus()
        {       lblSourceMobileHost.setText("Source MobileHost: "+sourceMobileHost);
                lblCurrentMobileHost.setText("Current MobileHost: "+currentMobileHost);
                lblCurrentMyChannel.setText("Current Channel: "+currentMyChannel);
                //draw source mobilehost in different color
                if(sourceMobileHost!=-1)
                {
                        int
tx=MapCells[map.getCellIDOfMobileHost(sourceMobileHost)].getMobileHostX(map.getMo
bileHostIndexInCell(sourceMobileHost));
                        int
ty=MapCells[map.getCellIDOfMobileHost(sourceMobileHost)].getMobileHostY(map.getMo
bileHostIndexInCell(sourceMobileHost));
                        g.setColor(new Color(0,255,0));
                        g.drawOval(tx,ty,radius,radius);
                }
                //display channel status
                if(currentMyChannel!=-1)
                {
                        int
tStatus=MapCells[map.getCellIDOfMyChannel(currentMyChannel)].getMyChannelStatus(m
ap.getMyChannelIndexInCell(currentMyChannel));
                        lblCurrentMyChannel.setText("Current Channel:
"+currentMyChannel+" ("+ChannelStatus.StatusString[tStatus]+")");
                }
        }
        //mouse motion listener methods
        public void mouseDragged(MouseEvent me)
```

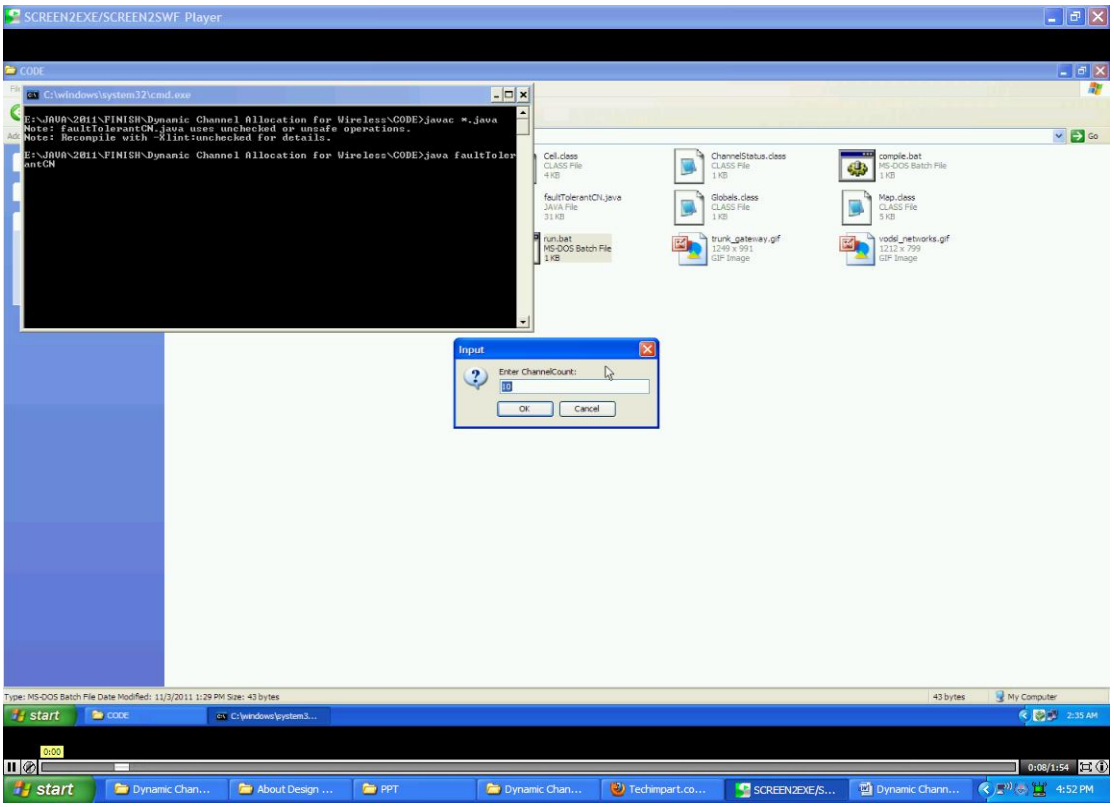
```
{                   //
}
public void mouseMoved(MouseEvent me)
{       int xposition=me.getX();
        int yposition=me.getY();
        lblXY.setText("("+xposition+","+yposition+")");
        int n=map.getTotalMobileHostsCount();
        int trad=radius-2;
        //check currentmobilehost
        for(int i=0;i<map.getCellCount();i++)
        {       for(int j=0;j<MapCells[i].getMobileHostCount();j++)
                {       int x1=MapCells[i].getMobileHostX(j)-trad;
                        int x2=MapCells[i].getMobileHostX(j)+trad;
                        int y1=MapCells[i].getMobileHostY(j)-trad;
                        int y2=MapCells[i].getMobileHostY(j)+trad;
                        if(between(xposition,yposition,x1,y1,x2,y2)==true)
                        {
currentMobileHost=MapCells[i].getMobileHostID(j);
                                displayStatus();
                        }
                }
        }
        //check currentmychannel
        for(int i=0;i<map.getCellCount();i++)
        {       for(int j=0;j<MapCells[i].getMyChannelCount();j++)
                {       int x1=MapCells[i].getMyChannelX(j)-trad;
                        int x2=MapCells[i].getMyChannelX(j)+trad;
                        int y1=MapCells[i].getMyChannelY(j)-trad;
                        int y2=MapCells[i].getMyChannelY(j)+trad;
                        if(between(xposition,yposition,x1,y1,x2,y2)==true)
                        {
currentMyChannel=MapCells[i].getMyChannelID(j);
                                displayStatus();
                        }
                }
        }
}


//mouselistener methods
public void mouseClicked(MouseEvent me)
{       if(currentMobileHost>=0)
        {       if(sourceSelected==false)
                {       sourceMobileHost=currentMobileHost;
                        sourceSelected=true;
                }       displayStatus();
        }
        //change channel status if right-clicked
        int mask=InputEvent.BUTTON1_MASK-1;
        int mods=me.getModifiers()&mask;
        if(mods==4) //right-clicked
        {       if(currentMyChannel>=0)
                {       swapChannelStatus(currentMyChannel);
                }
```

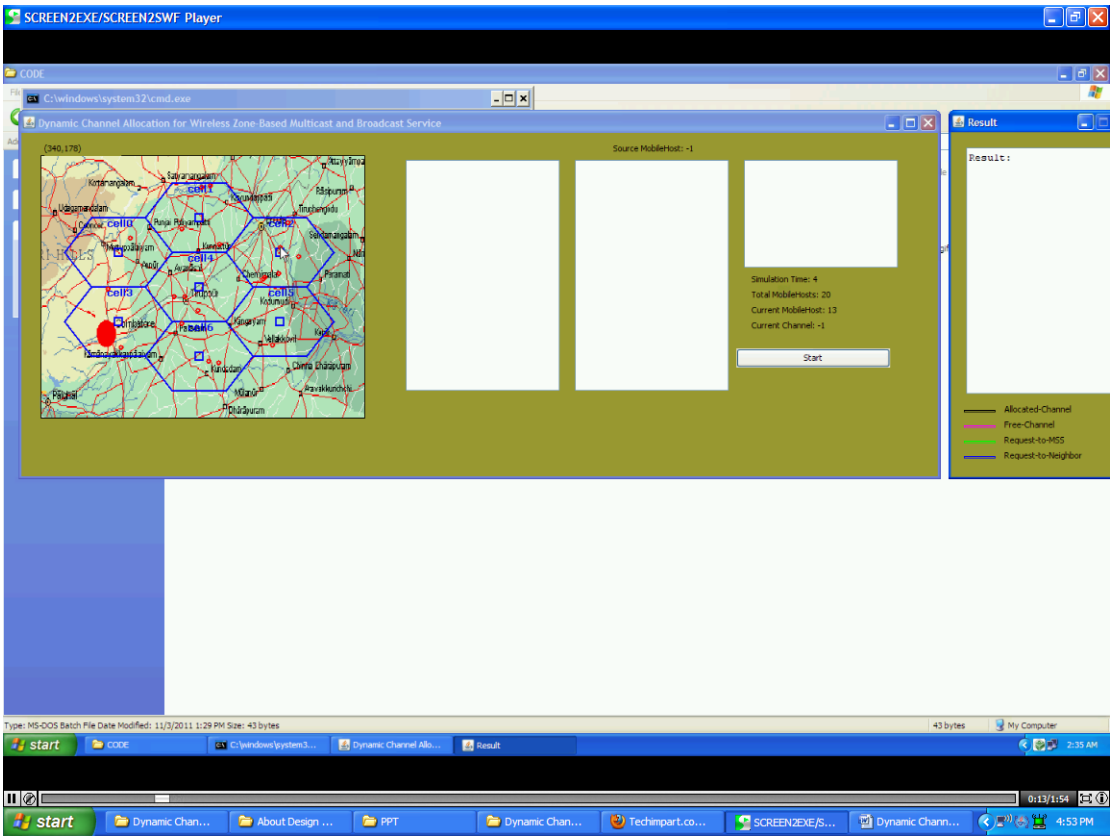
```
        }
}
public void mouseEntered(MouseEvent me)
{               //
}
public void mouseExited(MouseEvent me)
{               //
}
public void mousePressed(MouseEvent me)
{               //
}
public void mouseReleased(MouseEvent me)
{               //
}

//methods
void addResultText(String tStr)
{       txtResult.setText(txtResult.getText()+tStr+".");
}
void addRandomMobileHosts(int tCount)
{       for(int i=0;i<tCount;i++)
        {       try
                {       double random=java.lang.Math.random();
                        int no=(int)(map.getCellCount()*random);
                        if(map.getTotalMobileHostsCount()==0) no=4;
                        double random1=java.lang.Math.random();
                        int no1=(int)(((double)Globals.HexWidth-5.0)*random1);
                        double random2=java.lang.Math.random();
                        int no2=(int)(((double)Globals.HexHeight-5.0)*random2);
                        Graphics2D g2d=(Graphics2D)g;
                        g2d.setStroke (new BasicStroke (2.0f));
                        g.setColor(new Color(255,0,0));
                        int xposition=MapCells[no].getx()+no1;
                        int yposition=MapCells[no].gety()+no2;
                        g.drawOval(xposition,yposition,radius,radius);
                        MapCells[no].addMobileHost(xposition,yposition);
                }catch(Exception ex)
                {       System.out.println ("Error: "+ex.getMessage());
                }
        }
}
void addRandomMyChannels(int tCount)
{       for(int i=0;i<tCount;i++)
        {       try
                {       double random=java.lang.Math.random();
                        int tCellID=(int)(map.getCellCount()*random);
                        if(map.getTotalMyChannelsCount()==0) tCellID=4;
                        addNewChannelToCell(tCellID);
                }
                catch(Exception ex)
                { System.out.println("Error: "+ex.getMessage());
                } } }
void addNewChannelToCell(int CellID)
{       double random1=java.lang.Math.random(); //random x
```

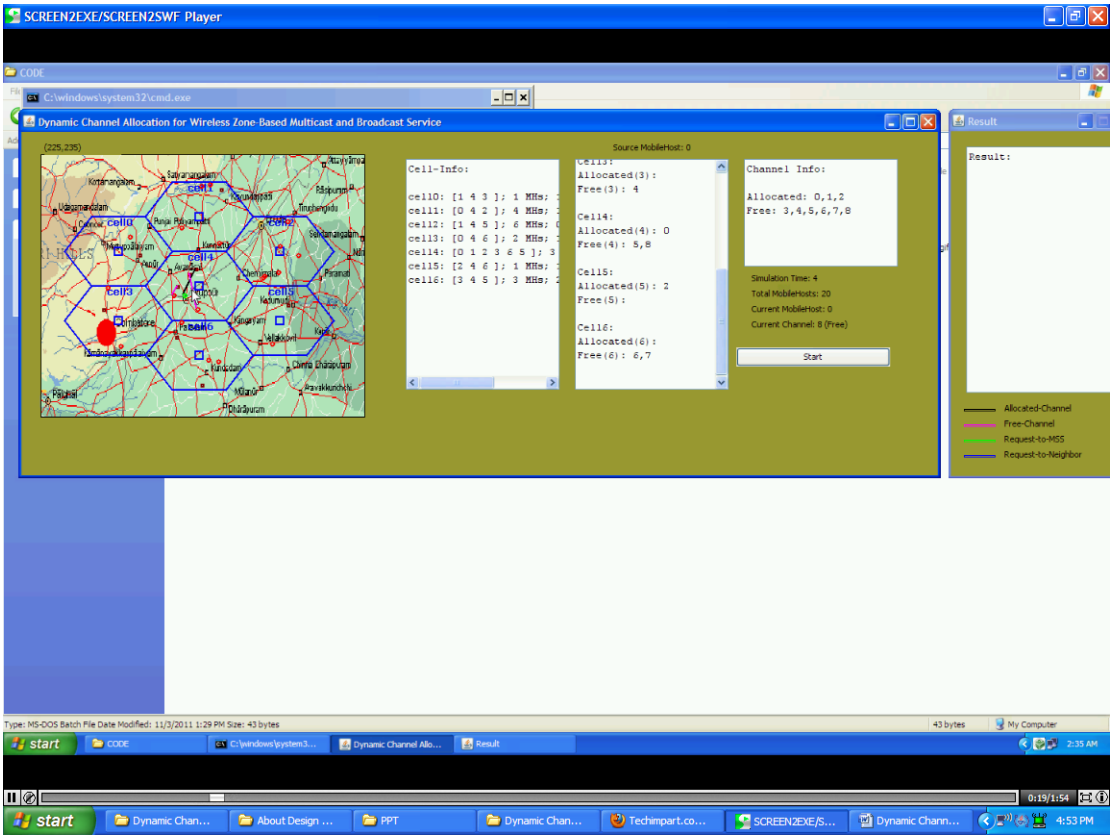
```
                int no1=(int)(((double)Globals.HexWidth-5.0)*random1);
                double random2=java.lang.Math.random(); //random y
                int no2=(int)(((double)Globals.HexHeight-5.0)*random2);
                double random3=java.lang.Math.random(); //random status
                int no3=(int)(2*random3);
                int tx=MapCells[tCellID].getx()+no1;
                int ty=MapCells[tCellID].gety()+no2;
                if(MapCells[tCellID].addMyChannel(tx,ty,no3)==true)
                {       g.setColor(new Color(128,0,0));
                        g.drawRect (tx, ty,3,3);
                }
        }
        public static void main(String args[])
        {       try {
UIManager.setLookAndFeel ("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
                } catch (Exception ex) {
                        System.out.println ("Failed loading L&F: ");
                        System.out.println (ex);
                        ex.printStackTrace ();
                } new faultTolerantCN ();
                }
        }
```

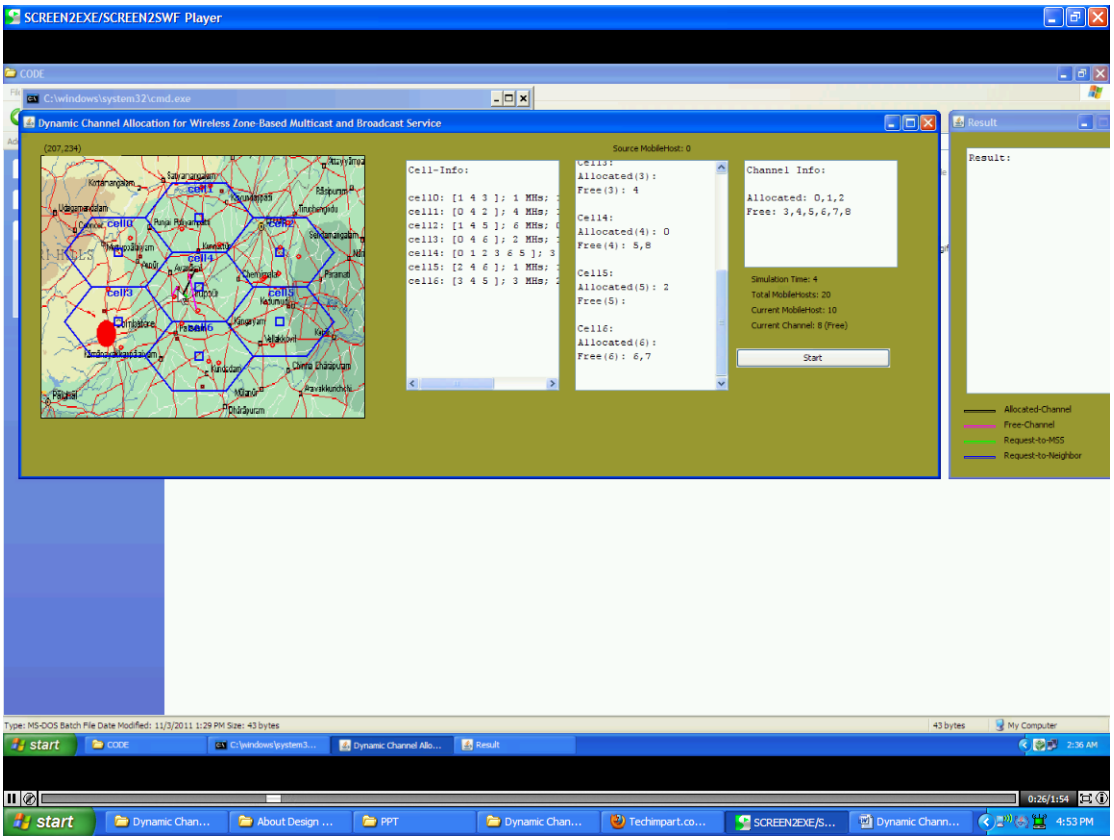
7. SCREENS

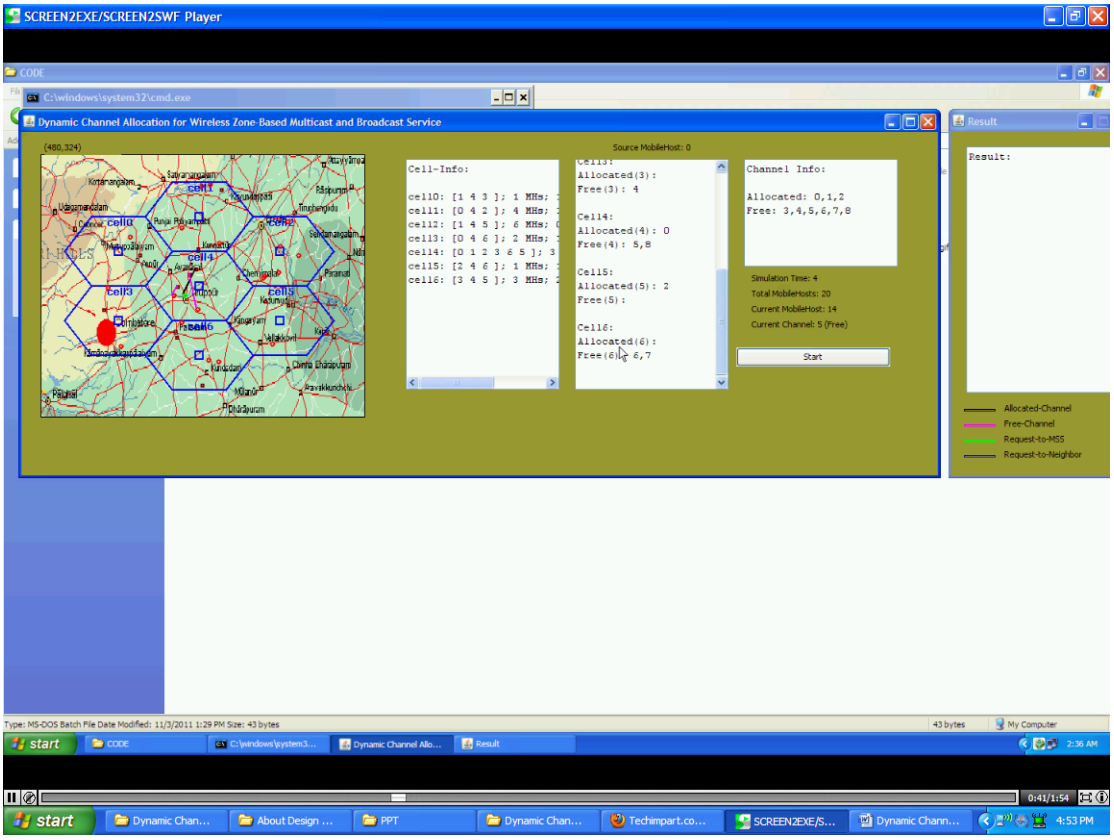

SCREEN2EXE/SCREEN2SWF Player
C:\windows\system32\cmd.exe
E:\JAVA\2011\FINISH\Dynamic Channel Allocation for Wireless\CODE>javac *.java
Note: FaultTolerantCN.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
E:\JAVA\2011\FINISH\Dynamic Channel Allocation for Wireless\CODE>java faultTolerantCN
Input
Enter ChannelCount:
OK
Cancel

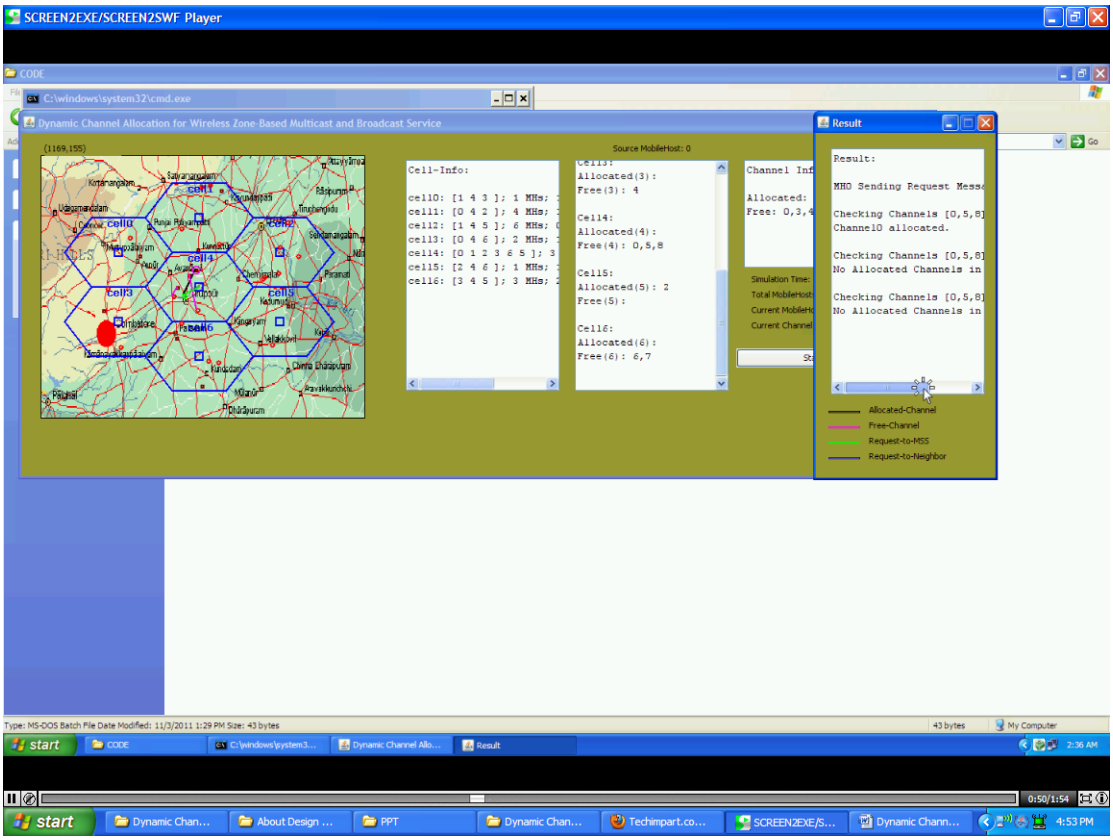

SCREEN2EXE/SCREEN2SWF Player
C:\windows\system32\cmd.exe
E:\JAVA\2011\FINISH\Dynamic Channel Allocation for Wireless\CODE>javac *.java
Note: FaultTolerantCN.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
E:\JAVA\2011\FINISH\Dynamic Channel Allocation for Wireless\CODE>java faultTolerantCN
Input
Enter ChannelCount:
OK
Cancel

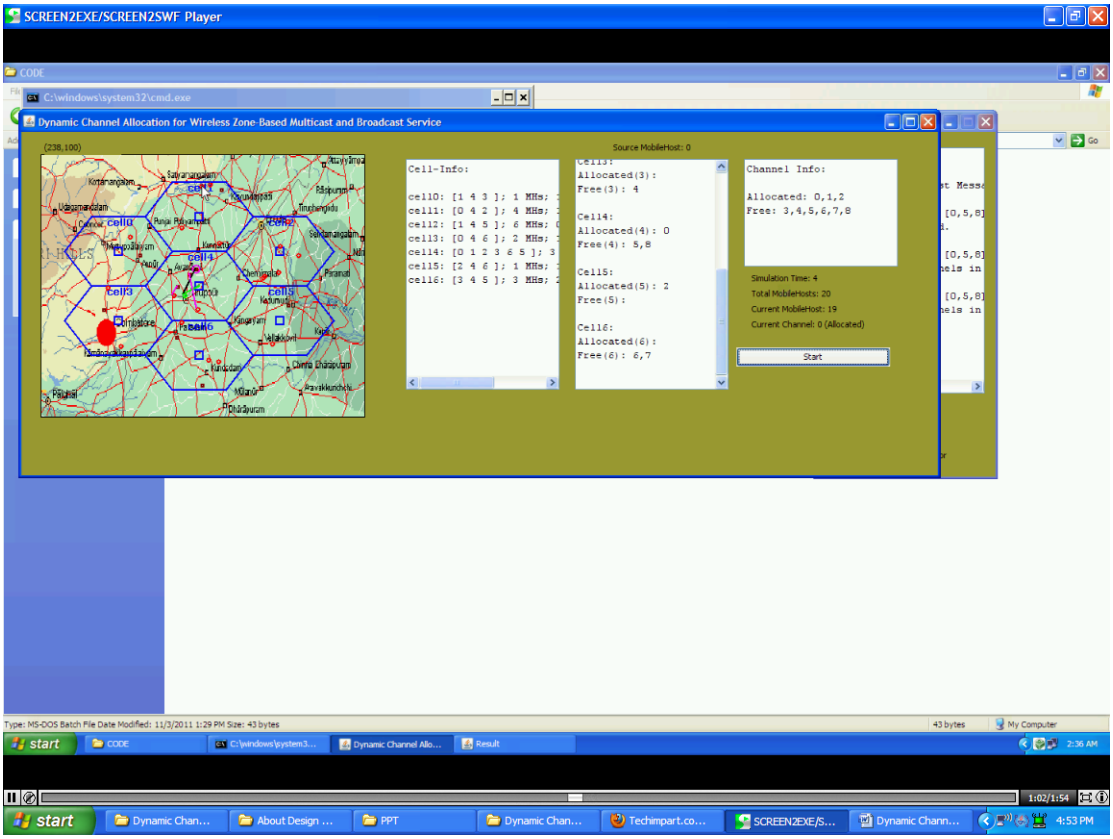

SCREEN2EXE/SCREEN2SWF Player
Dynamic Channel Allocation for Wireless Zone-Based Multicast and Broadcast Service
Result
Result:
Start
Allocated-Channel
Free-Channel
Request-to-MSS
Request-to-Neighbor

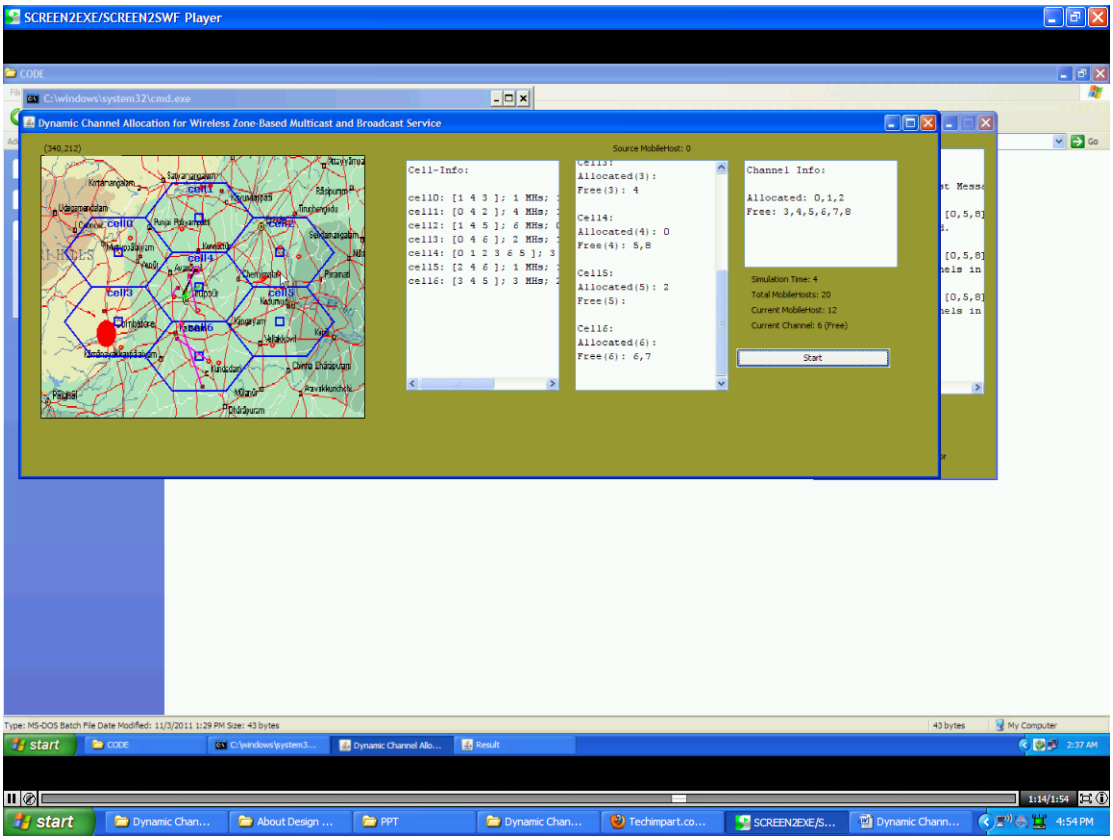

SCREEN2EXE/SCREEN2SWF Player
Dynamic Channel Allocation for Wireless Zone-Based Multicast and Broadcast Service
Result
Result:
Start
Allocated-Channel
Free-Channel
Request-to-MSS
Request-to-Neighbor

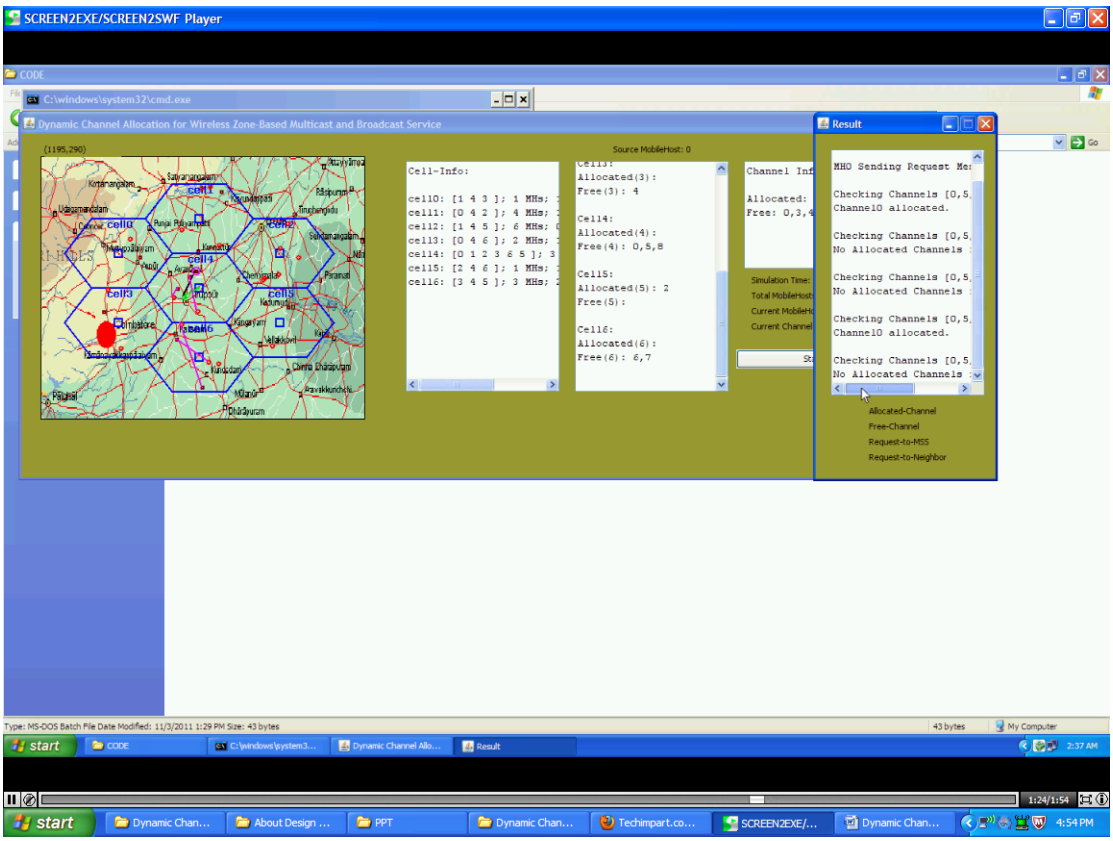

SCREEN2EXE/SCREEN2SWF Player
Dynamic Channel Allocation for Wireless Zone-Based Multicast and Broadcast Service
Cell-Info:
Channel Info:
Allocated: 0,1,2
Free: 3,4,5,6,7,8
Start
Result:

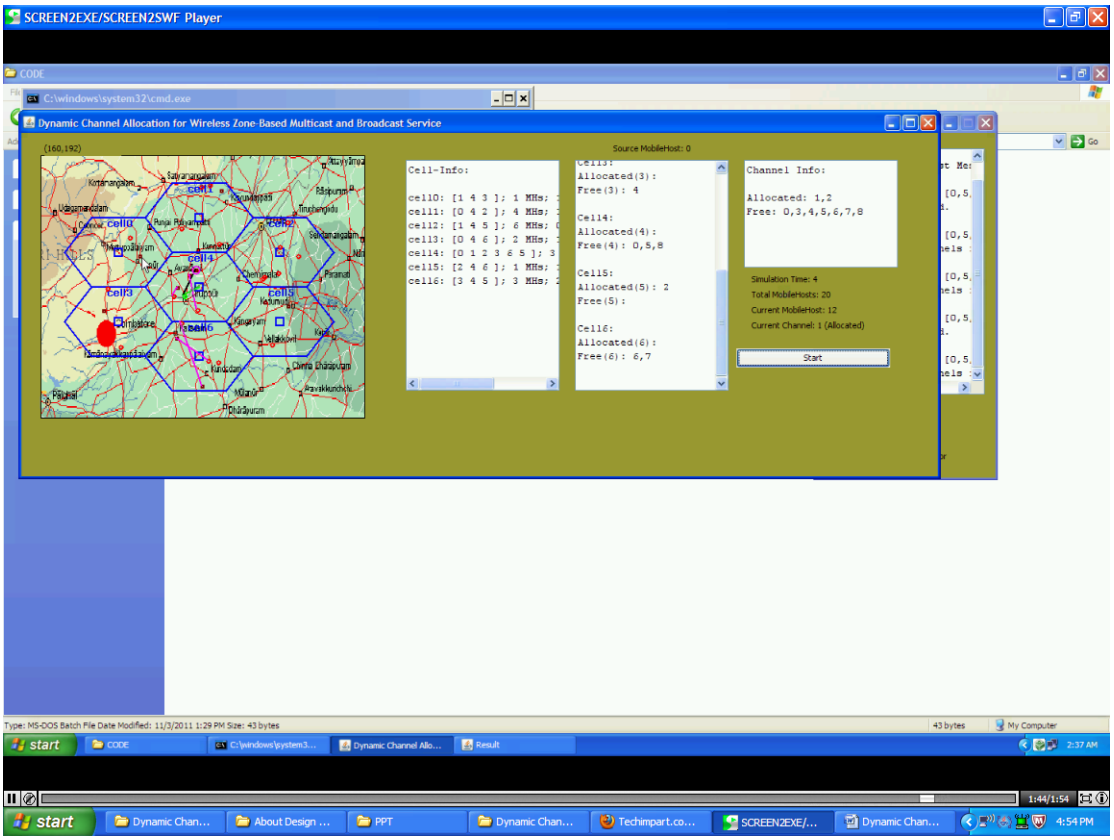

SCREEN2EXE/SCREEN2SWF Player
Dynamic Channel Allocation for Wireless Zone-Based Multicast and Broadcast Service
Cell-Info:
Channel Info:
Allocated: 0,1,2
Free: 3,4,5,6,7,8
Start
Result:

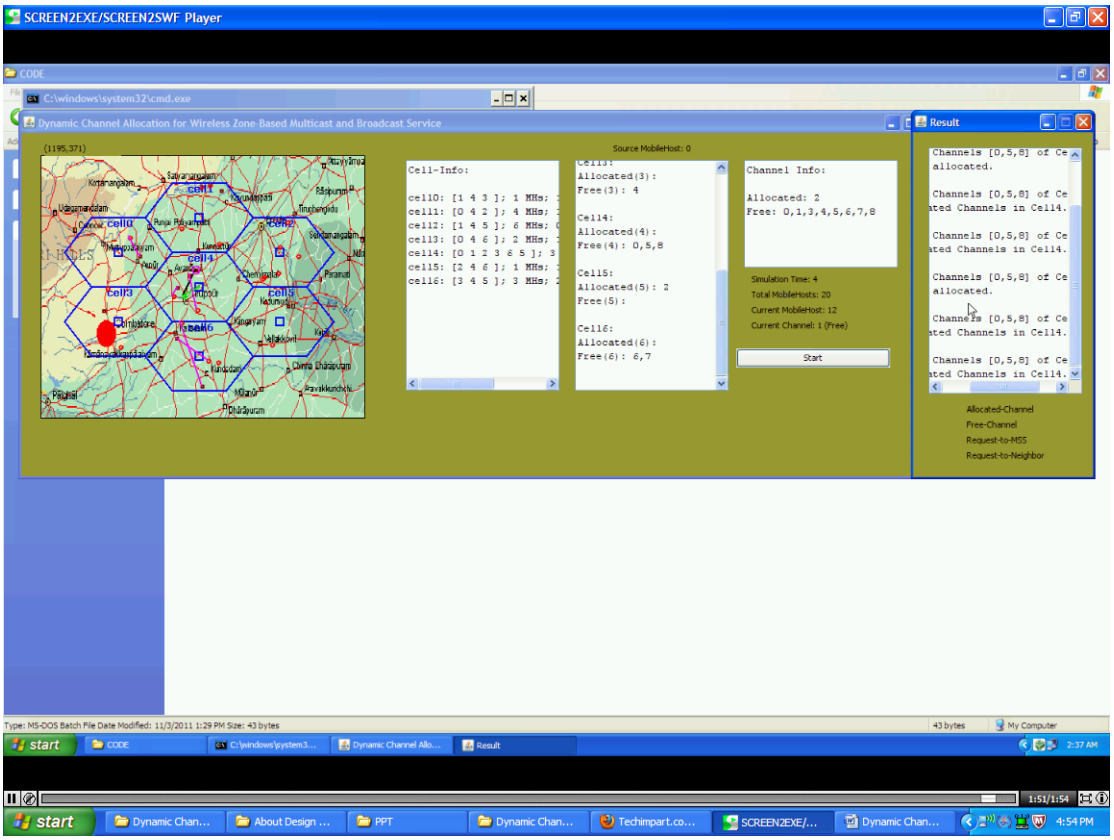

SCREEN2EXE/SCREEN2SWF Player
Dynamic Channel Allocation for Wireless Zone-Based Multicast and Broadcast Service
Result
Cell-Info:
Result:
Allocated-Channel
Free-Channel
Request-to-MSS
Request-to-Neighbor


SCREEN2EXE/SCREEN2SWF Player
Dynamic Channel Allocation for Wireless Zone-Based Multicast and Broadcast Service
Cell-Info:
Channel Info:
Allocated: 0,1,2
Free: 3,4,5,6,7,8
Simulation Time: 4
Total MobileHosts: 20
Current MobileHost: 19
Current Channel: 0 (Allocated)
Start


SCREEN2EXE/SCREEN2SWF Player
Dynamic Channel Allocation for Wireless Zone-Based Multicast and Broadcast Service
Cell-Info:
Channel Info:
Allocated: 0,1,2
Free: 3,4,5,6,7,8
Start


SCREEN2EXE/SCREEN2SWF Player
Dynamic Channel Allocation for Wireless Zone-Based Multicast and Broadcast Service
Result
Cell-Info:


SCREEN2EXE/SCREEN2SWF Player
Dynamic Channel Allocation for Wireless Zone-Based Multicast and Broadcast Service
Cell-Info:
Channel Info:
Allocated: 1,2
Free: 0,3,4,5,6,7,8
Start


SCREEN2EXE/SCREEN2SWF Player
CODE
C:\windows\system32\cmd.exe
Dynamic Channel Allocation for Wireless Zone-Based Multicast and Broadcast Service
Result
Cell-Info:
Channel Info:
Allocated: 2
Free: 0,1,3,4,5,6,7,8
Simulation Time: 4
Total MobileHosts: 20
Current MobileHost: 12
Current Channel: 1 (Free)
Start
Allocated-Channel
Free-Channel
Request-to-MSS
Request-to-Neighbor
start
Dynamic Chan...
About Design ...
PPT
Techimpart.co...
SCREEN2EXE/...
4:54 PM

8. CONCLUSION

T his paper proposed DCA and EDCA for more flexible channel allocation for MBS, which improve the performance for users. However, more signaling overheads are caused by DCA and EDCA. Therefore, we proposed analytical and simulation models to study the Sf performance for Basic, DCA, and EDCA. Our study provides the following guidance for MBS network operators when to use Basic, DCA, or EDCA:

• When the macro diversity functions well (i.e., d_z is small), Basic is the good choice since the Sf performance for the three schemes are almost the same. On the other hand, when dz is large, EDCA is the better choice.

• As the number of MBS calls in a cell is larger (i.e., higher AO""i' lower 'T/i, or lower f.Lm), EDCA is the better choice.

• The performance enhancement of EDCA over DCA decreases as AO\, i increase. Therefore, when traditional call arrival rate AO\, i is higher enough, DCA is suggested.

• The performance trend for the three schemes for different MS mobility patterns are almost the same.

9. FUTURE ENHANCEMENT

The system Dynamic Channel allocation for wireless zone based Multicast an Broadcast Service was designed under the Mobile Computing Networking, now it good for IGMP-MBS CONTENT transformation from one MBS zone BS to another BS which belongs to same or different zones.

In future we may modify the system for 3G/4G MBS content transformation by mobile Computing.

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

3. Mobile Computing

[1] IEEE. IEEE Standard for Local and Metropolitan Area Networks Part 16: Air Interface for Broadband Wireless Access Systems. IEEE Std 802.16-2009, May 2009.

[2] IEEE. IEEE 802. I 6m System Description Document. IEEE STD 802.16m, December 2008.

[3] WiMAX Forum. WiMAX Forum Network Architecture Release 1.5 Versions I Draft 0 - Stage 3: Detailed Protocols and Procedure. November 2009.

[4] Wang, J., Venkatachalam, M., and Fang, Y System Architecture and Cross-Layer Optimization of Video Broadcast over WiMAX. IEEE Journal on Selected Areas in Communications, 25(4):712-721, May 2007.

[5] Jeng, J.-Y, and Lin, Y.-B. Equal Resource Sharing Scheduling for PCS Data Services. ACM Wireless Networks, 5(1):41-55, January 1999.

[6] Lin, Y-B., and Yang, S.-R. A Mobility Management Strategy for GPRS. IEEE Transactions on Wireless Communications, 2(6): 1178-1188, November 2003.

[7] Lin, P. and Lin, Y-B. Channel Allocation for GPRS. IEEE Transactions on Vehicular Technology, 50(2):375-387, March 2001.

[8] Lin, P. Channel Allocation for GPRS with Buffering Mechanisms. ACM Wireless Networks, 9(5):431-441, September 2003.

[9] Etemad, K., and Wang, L. Multicast and Broadcast Multimedia Services in Mobile WiMAX Networks. IEEE Communications Magazine, 47(10):84--91, October 2009.

[10] Lai, y-c., Lin, P., Fang Y, and Chen, W.-H. Channel Allocation for UMTS Multimedia Broadcasting and Multicasting. IEEE Transactions on Wireless Communications, 7(11):4375-4383, November 2008.

Sites Referred:

http://java.sun.com

http://www.sourcefordgde.com

http://www.jfree.org/

http://www.networkcomputing.com/