

A Project Report on

Distributed Data over Network Systems

*submitted in the partial fulfillment of the requirements
for the award of the degree of*

Bachelor of Engineering
in
INFORMATION TECHNOLOGY

by

Niranjan K. Chavan, Suhas D. Mane, Omkar S. Pathak, Shrimant B. Bandgar,
Nishantkumar Sinha

under the guidance of

Mr. T. B. Patil



DEPARTMENT OF INFORMATION TECHNOLOGY

KIT'S College of Engineering, Kolhapur (Maharashtra)

2009 - 2010

Declaration

We, the undersigned, hereby declare that the Project work entitled '**Distributed Data over Network Systems**' submitted by us to KIT's College of Engineering, Kolhapur, for the award of the degree of Bachelor of Engineering in Information Technology, under the guidance of **Mr. T. B. Patil**, is our original work.

We further declare that to the best of our knowledge and belief, this work has not been submitted to this or any other university.

Niranjan K. Chavan

Suhas D. Mane

Omkar S. Pathak

Shrimant B. Bandgar

Nishantkumar Sinha

Certificate

This is to certify that, **Mr. Niranjan K. Chavan, Mr. Suhas D. Mane, Mr. Omkar S. Pathak, Mr. Shrimant B. Bandgar, Mr. Nishantkumar Sinha** has satisfactorily completed Project work on '**Distributed Data over Network Systems**' in partial fulfillment of the requirements for the degree of '**Bachelor of Engineering**' in Information Technology.

This Project work is a record of student's own work carried out by them during the academic year 2009-2010, under the guidance of **Mr. T. B. Patil**.

(Mr. T. B. Patil)

Guide

Dept. of Information Technology

(Prof. Mr. A. S. Patil)

Head

Dept. of Information Technology

Place: KIT's College of Engineering, Kolhapur.

Date:

Acknowledgement

Every orientation work has an imprint of many people and it becomes our duty to express deep gratitude for the same. During preparation for this Project work, We received endless help from a number of people and feel that this report would be incomplete if we do not convey grateful thanks to them. This acknowledgment is a humble attempt to thanks all those who were involved in this project work.

First and foremost, We take the opportunity to thank our guide **Mr. T. B. Patil** for giving us this opportunity to carry on this Project work, excellent encouragement and constant guidance throughout this project work. We also thank **Prof. Mr. A. S. Patil**, Head of the Department (Information Technology), for his indispensable support, priceless suggestions and for valuable time.

We specially would like to thank our classmates, friends and well wishers for motivating us in one of the most difficult period of our life. We also acknowledge those who helped us directly and indirectly in many ways in completion of this Project work.

We are forever indebted to our parents for their love, moral support and encouragement throughout our education.

Niranjan K. Chavan

Suhas D. Mane

Omkar S. Pathak

Shrimant B. Bandgar

Nishantkumar Sinha

Abstract

The idea behind the system is to allow a group of computers (nodes) to be combined into a single network. The system should allow mass storage. The idea for the project came about after constantly running out of space on a personal computer, there are a lot of large capacity systems on the Internet or LAN and in organizations which are not being efficiently used.

In a concern, computers are grouped together to form a network. The architecture for the system is fairly straightforward; the network would be controlled by a single manager (the Central manager [CM]). The CM would keep a central index for all the files in the system. Requests from a client would be searched against the list and the CM can send a request that the node act on the file.

The CM would not give which node the file is on, but that the file exists and can be accessed; giving the end user has the correct permissions. The CM would act as a **‘Virtual File system’** in the Network.

Contents

1	Distributed Data over Network Systems	1
1.1	Introduction	1
1.2	Functionalities	2
1.2.1	DDNS Central Manager (DDNS_CM)	2
1.2.2	DDNS Node (DDNS_NODE)	3
1.2.3	DDNS Client (DDNS_CLIENT)	3
2	Requirement Specification	4
2.1	Introduction	4
2.1.1	Purpose	4
2.1.2	Scope	4
2.1.3	Conventions	4
2.1.4	References	5
2.1.5	Document Overview	5
2.2	Overall Description	5
2.2.1	Product Perspective	5
2.2.2	Product Features	6
2.2.3	Operating Environment	6
2.2.4	General Constraints	6
2.2.5	Assumptions & Dependancies	7
2.3	Engineering Requirements	7
2.3.1	CSCI Capability Requirements	7

2.3.2	Configuration (DDNS_CM_CAP_001)	7
2.3.3	Handle Client Inputs (DDNS_CM_CAP_002)	7
2.3.3.1	User Login	7
2.3.3.2	User Logout	8
2.3.3.3	Data Commit	8
2.3.3.4	Read Data	9
2.3.3.5	Delete Data	9
2.3.3.6	Directory Listing	9
2.3.4	Handle Node Inputs (DDNS_CM_CAP_003)	9
2.3.4.1	Free Space Response	9
2.3.4.2	Current Operation Completion	10
2.3.5	User Database (DDNS_CM_CAP_004)	10
2.3.6	Database (DDNS_CM_CAP_005)	10
2.3.7	Master File List Database (DDNS_CM_CAP_006)	11
2.4	Project Schedule	11
3	System Design	13
3.1	The Process Model	13
3.1.1	The Spiral Model	13
3.2	Unified Modelling Language Diagrams	14
3.2.1	Class Diagram	14
3.2.2	Sequence Diagram	16
3.2.3	Use Case Diagram	16
3.2.3.1	Use Case	16
3.2.4	Collaboration Diagram	16
3.2.5	Activity Diagram	19
3.2.6	Deployment Diagram	20
4	Implementation	22
4.1	Remote Method Invocation	22
4.1.1	Different Ways To Do It	24

4.1.1.1	RMI Interface	24
4.1.1.2	RMI Server Implementation	25
4.1.1.3	RMI Clients	26
4.1.1.4	Compiling and Running	27
4.2	Advantages of RMI	28
5	Working with Distributed Data over Network System (DDNS)	31
5.1	Introduction	31
5.2	DDNS_NODE	31
5.3	DDNS_CM	33
5.3.1	DDNS Server Admin	36
5.3.2	DDNS Server	40
5.4	DDNS_CLIENT	41
6	Conclusion	48

List of Figures

1.1	External Interface Diagram	2
3.1	The Spiral Model	14
3.2	Class Diagram	15
3.3	Sequence Diagram	17
3.4	Use Case Diagram	18
3.5	Collaboration Diagram	19
3.6	Activity Diagram	20
3.7	Deployment Diagram	21
4.1	RMI Basic Architecture	23
5.1	Starting Node	32
5.2	RMI Registry	32
5.3	Directory creation and File Storing	33
5.4	Directory creation and File Storing on Linux Node	34
5.5	Table : Users	35
5.6	Table : Node	35
5.7	Table : File system	36
5.8	Admin Login	37
5.9	Main Screen	37
5.10	Add or Remove User	38
5.11	Add Node	39
5.12	Choose drive	40

5.13	Store File System	41
5.14	Starting DDNS Server	42
5.15	Client Login	42
5.16	File System Hierarchy	43
5.17	Thread Created	43
5.18	connection successfull	44
5.19	Getting started using RMI	44
5.20	Cascade	45
5.21	Tile	46
5.22	File is Busy	46
5.23	Multiple files opened	47

Chapter 1

Distributed Data over Network Systems

1.1 Introduction

The idea behind the system is to allow a group of computers (nodes) to be combined into a single network. The system should allow mass storage. The idea for the project came about after constantly running out of space on a computer, there are a lot of large capacity systems on the Internet and in organizations which are not being efficiently used.

The architecture 1.1 for the system is fairly straight forward; the network would be controlled by a single manager (the Central manager [CM]). The CM would keep a central index for all the files in the system. The CM would give which node the file is on and can be accessed; giving the user has the correct permissions. The CM would act as a **‘Virtual File system’**.

The Goals for the system are:

- Incorporate encryption for the files, which would allow remote users to access private files Across the net.
- For every file that exists there will be a duplicate for backup purposes, this would allow RAID Type protection if a node crashed or the connection died.
- Portable system that can be moved over to Linux as well as under Microsoft Windows.

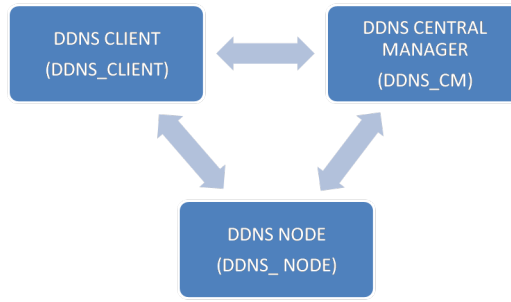


Figure 1.1: External Interface Diagram

1.2 Functionalities

1.2.1 DDNS Central Manager (DDNS_CM)

- Authenticates the DDNS_CLIENT.
- Receives read / write file requests from DDNS_CLIENT.
- DDNS_CM searches the catalog for requested file, and gives the identification of node having the requested file on a network to the DDNS_CLIENT.
- Receives the data commit (in case of write operation) acknowledgement from DDNS_NODE and forwards it to DDNS_CLIENT.
- Sends ping request, to ascertain the speed of the connection between the DDNS_NODE and DDNS_CM.
- Sends periodically free space request, to get the total free space on the DDNS_NODES.
- Receives node connect request, this is when a node wants to connect to the network, if it is a new node then a new node entry will be created in catalog at DDNS_CM.

1.2.2 DDNS Node (DDNS_NODE)

- Receives read / write file request from DDNS_CLIENT.
- Performs the read / write operation and sends the acknowledgement to the DDNS_CM.
- Sends ping response, this will return a response to the ping request, giving node connection speed.
- Sends free space response, giving the free space on the DDNS_NODE to the DDNS_CM.
- Sends node connect request, this is when a node wants to connect to the network, if it is a new node then a new node entry will be created.

1.2.3 DDNS Client (DDNS_CLIENT)

- Sends read / write request to the DDNS_CM and it will receive DDNS_NODE having requested file.
- Sends read / write request to the DDNS_NODE.
- Receives the data commit acknowledgement from the DDNS_CM.

Chapter 2

Requirement Specification

2.1 Introduction

2.1.1 Purpose

The purpose of this document is to present a detailed description of the Distributed Data Over Network System. It will explain the purpose and features of the system, the interfaces of the system, what the system will do, the constraints under which it must operate and how the system will react to external stimuli. This document is intended for the System Administrator.

2.1.2 Scope

The idea behind the system is to allow a group of computers (nodes) to be combined into a single network. The system should allow mass storage. The idea for the project came about after constantly running out of space on a computer, there are a lot of large capacity systems on the Internet and in organizations which are not being efficiently used.

2.1.3 Conventions

This Software Requirements Specification (SRS) concerns the Central Node Manager (DDNS_CM) Computer Software Configuration Item (CSCI) of the Distributed Data Network System identified in the table below:

Item	Abbrevation	Meaning
SYSTEM	DDNS	Distributed Data Network System
CSCI	DDNS_CM	DDNS Central Manager
	DDNS_NODE	DDNS Data Store Node
	DDNS_CLIENT	DDNS Client who wish to access file system

Table 2.1: Identification

2.1.4 References

- Parallel Virtual File System implemented by Clemson University. (www.clemson.edu and www.pvfs.org).
- Google File System.

2.1.5 Document Overview

This document specifies the requirements to be satisfied by the CSCI. The interfaces of the CSCI with the other configuration items, the capabilities to be performed by the CSCI, the timing and sizing requirements, the requirements for design, security factors, the requirements for the qualification of the CSCI.

This specification forms a reference document for the CSCI design. It is also intended for use during the drafting of the qualification testing documents and in performing the final qualification, assuring that the CSCI conforms to the requirements specified in this RS.

2.2 Overall Description

2.2.1 Product Perspective

DDNS aims at allowing the System Administrator to uniformly distribute a local file system over a DDNS_NODES to overcome 'continuously run out of space' critic and provides the multiple clients or users of the system to access this file system with 'location transparency'. The DDNS_CM acts as a 'virtual' file system.

2.2.2 Product Features

The DDNS_CM provides a node and data manager for Distributed Data Network System project; it will interact the rest of the DDNS System and manage the network. The DDNS_CM CSCI achieves the following functions:

- Handles data write requests to the DDNS from a client.
- Handles data read requests from the DDNS to a client.
- Handles user login/logout requests from a client.
- Manages the master user database, including adding, deleting and modifying users.
- Manages the master node database, including adding and deleting of nodes.
- Handles messages from a node, including connecting, disconnecting, adding a new node, file request responses.

2.2.3 Operating Environment

- Language : Java.
- Platform : Windows XP, Linux Operating System.
- Hardware : PC (P IV).
- Database : Oracle.

2.2.4 General Constraints

Each user must keep his / her password confidential. Moreover the user must have individual ID for creating a login. Only DDNS_CM can control DDNS_CLIENT and DDNS_NODE addition and deletion.

Capability Name	Capability Identifier
Configuration	DDNS_CM_CAP_001
Handle Client Inputs	DDNS_CM_CAP_002
Handle Node Inputs	DDNS_CM_CAP_003
User Database	DDNS_CM_CAP_004
Node Database	DDNS_CM_CAP_005
Master File List Database	DDNS_CM_CAP_006

Table 2.2: CSCI Capability Requirement

2.2.5 Assumptions & Dependancies

Every DDNS_NODE is having dedicated free space available on hard disk. Both DDNS_CM and DDNS_NODE are continuously active.

2.3 Engineering Requirements

2.3.1 CSCI Capability Requirements

2.3.2 Configuration (DDNS_CM_CAP_001)

- **DDNS_CM_RS001** : The CM shall be able to be configured to use any port between 1024 and 32000, if an invalid port number is specified 7777 will be used by default.
- **DDNS_CM_RS002** : The DDNS_CM shall be configurable to enable/disable logging of messages to a log file. The setting shall be read in from a configuration file at initialization.
- **DDNS_CM_RS003** : The interval between a checkpoint shall be in the range of 20 minutes to 72 hours.

2.3.3 Handle Client Inputs (DDNS_CM_CAP_002)

2.3.3.1 User Login

- **DDNS_CM_RS004** : The CM shall handle a login from the client.

- **DDNS_CM_RS005** : Only one connection per user is permitted, if a second login is attempted it will automatically fail.
- **DDNS_CM_RS006** : A login request shall consist of the username and password for a user.
- **DDNS_CM_RS007** : If a user enters the wrong password x times (see configuration capabilities) then the account will be locked and can only be unlocked by contacting an admin.

2.3.3.2 User Logout

- **DDNS_CM_RS008** : The CM shall check that the logout request came from the same connection as the login.

2.3.3.3 Data Commit

- **DDNS_CM_RS009** : If the connection receiving the request has no valid user logged in then the request shall fail and “Permission Denied” returned to the client.
- **DDNS_CM_RS010** : A data file will contain physical data, be it binary or text and can be read/accessed using software.
- **DDNS_CM_RS011** : A directory is used to break down files down into more manageable parts.
- **DDNS_CM_RS012** : A filename in a directory shall be unique, unique means differ by name and not by case test.txt and tesT.TXT is classified as matched.
- **DDNS_CM_RS013** : The parent directory of the file being committed will be marked “in use”.
- **DDNS_CM_RS014** : If two or more users attempt to commit a file in a directory with the same name a “file being committed” error will be returned and commit aborted.

- **DDNS_CM_RS015** : If the file does not exist then “File Not Found” will be returned to the client and update aborted.

2.3.3.4 Read Data

- **DDNS_CM_RS015** : If the connection receiving the request has no valid user logged in then the request shall fail and “Permission Denied” returned to the client.
- **DDNS_CM_RS016** : A read data cannot be performed on a file that has been flagged as “file being used”.

2.3.3.5 Delete Data

- **DDNS_CM_RS017** : If the connection receiving the request has no valid user logged in then the request shall fail and “Permission Denied” returned to the client.
- **DDNS_CM_RS018** : If user requesting the delete is not the file owner or an administrator then a permission error is returned and delete aborted.
- **DDNS_CM_RS019** : After deleting the file / directory Database at DDNS_CM should be updated.

2.3.3.6 Directory Listing

- **DDNS_CM_RS020** : If the connection receiving the request has no valid user logged in then the request shall fail and “Permission Denied” returned to the client.

2.3.4 Handle Node Inputs (DDNS_CM_CAP_003)

2.3.4.1 Free Space Response

- **DDNS_CM_RS021** : The free space response shall consist of the node ID and the free space in bytes.

2.3.4.2 Current Operation Completion

- **DDNS_CM_RS022** : After serving the read / write request (completed / aborted), node should inform DDNS_CM about status of current operation performed.

2.3.5 User Database (DDNS_CM_CAP_004)

- **DDNS_CM_RS023** : The user database shall store the following information about a user:
 1. Username.
 2. Password.
 3. Account Status.(locked / unlocked).
 4. Connection state.
- **DDNS_CM_RS024** : Empty passwords shall not be permitted.

2.3.6 Database (DDNS_CM_CAP_005)

- **DDNS_CM_RS025** : The node database shall store the following information about a node:
 1. Nodes IP address.
 2. Node Port number.
 3. Total Disk Space.
 4. Used Space.
- **DDNS_CM_RS026** : “Node exists” shall be returned if a node already exists on the IP address.

2.3.7 Master File List Database (DDNS_CM_CAP_006)

- **DDNS_CM_RS027** : The master file list database shall store the following information about a file:
 1. File Name.
 2. Parent Directory.
 3. Total Size.
 4. Node ID the file resides on.
 5. File Owner.
 6. Date Last Modified.
- **DDNS_CM_RS028** : The master file list database shall decide on the physical node location by firstly checking there is enough space and then comparing a ping result on each node, using the fastest node.

2.4 Project Schedule

Proposed Work	100 %	Duration
Selecting Topic August 2009	5	August 2009
Approaching To Guide	5	August 2009
Submitting The Synopsis	10	September 2009
Requirement Analysis	10	September 2009
System Design	20	October 2009
Coding	30	December 2009- February 2009
Testing	10	March 2009
Documentation	10	April 2009

Table 2.3: Schedule

Chapter 3

System Design

3.1 The Process Model

3.1.1 The Spiral Model

The project development is based on the standard procedure and practices. The system life cycle is Thoroughly followed. All the latest project management techniques, development models, and testing techniques are being used for development of application. The following methodology is used for Application Value Management 3.1.

- **Communication** : The software development process starts with communication between client and server. According to waterfall model client must interact with CM for his requirements.
- **Planning** : It includes complete estimation (e.g. cost estimation of project) and scheduling (complete timeline chart for project development) and tracking.
- **Modeling** : It includes detail requirement analysis and project design (algorithm, flowchart).
- **Construction** : It includes coding and testing steps: Coding: Design details are implemented using appropriate programming language.
- **Testing** : Testing is carried out to check whether the flow of coding is correct, to check out the errors of program.

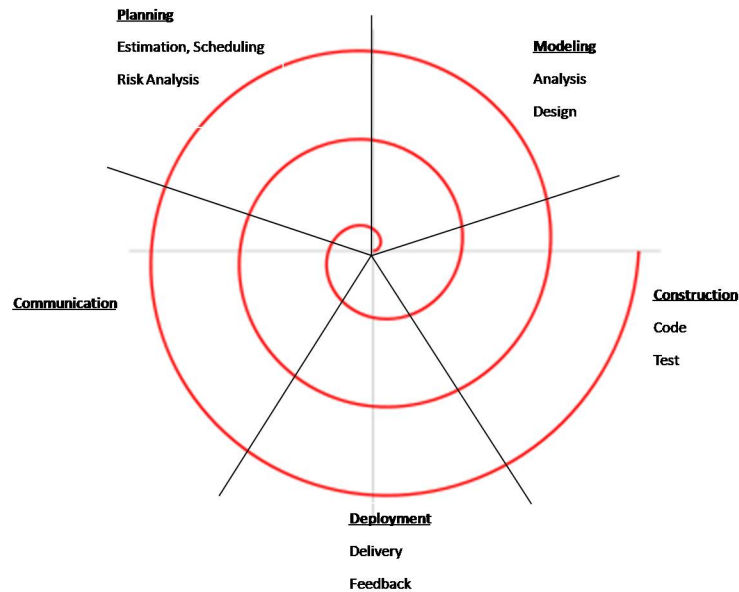


Figure 3.1: The Spiral Model

- **Deployment** : It includes implementation of a Distributed system over a LAN.

3.2 Unified Modelling Language Diagrams

3.2.1 Class Diagram

Class diagrams are the most common diagram found in modeling object oriented systems 3.2. A class diagram shows a set of classes, interfaces and collaboration and their relationships.

We use class diagram to model the static design view of a system. For the most part, this involves modeling a vocabulary of the system, modeling collaborations, or modeling schemas. Class diagrams are also the foundation for a couple of related diagrams: Component diagram and deployment diagrams. Class diagram are important not only for visualizing, specifying, and documenting structural models, but also for constructing executable system through forward and reverse engineering.

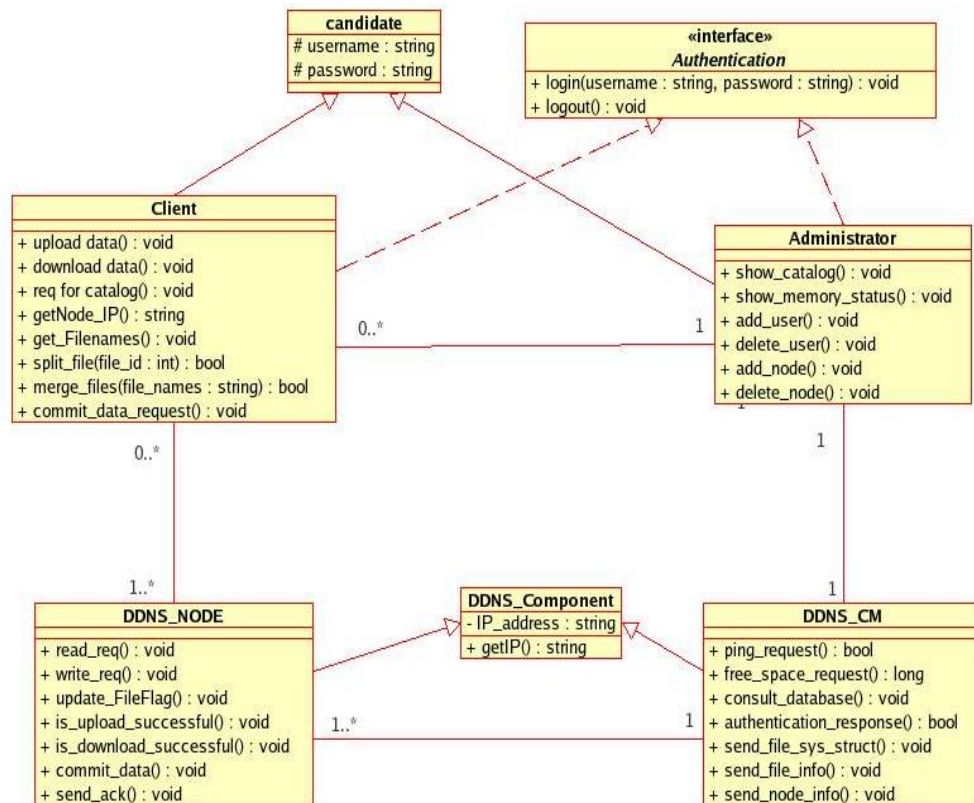


Figure 3.2: Class Diagram

3.2.2 Sequence Diagram

Each sequential element is arranged in horizontal sequence, with message passing back and forward between elements 3.3. Stereotyped elements, such as boundary, control and entity may be used to illustrate screens, controllers and database items, respectively.

3.2.3 Use Case Diagram

A use case diagram shows the relationship among actors and use cases within the system 3.4. A use case is a unique function in a system, and an actor is the person or software that performs the action or use case. For example, a client is the actor that performs the function (use case) of sending request to PC; the server is the actor that performs the use case of performing given request. In UML, systems are grouped into squares, actors are represented by stick figure, use cases are denoted by ovals, and the lines show how actors use the system.

3.2.3.1 Use Case

A use case is a scenario that describes the use of a system by an actor to accomplish a specific goal. What does this mean? An actor is a user playing a role with respect to the system. Actors are generally people although other computer systems may be actors. A scenario is a sequence of steps that describes the interactions between an actor and the system. The use case model consists of the collection of all actors and all use cases. Use cases help us to capture :

- The system's functional requirements from the user's perspective.
- Actively involve users in the requirements-gathering process.
- Provide the basic for identifying major classes and their relationship.
- Serve as the foundation for developing system test cases.

3.2.4 Collaboration Diagram

A collaboration diagram emphasizes the organization of the objects that participate in an interaction 3.5. Collaboration diagram are formed by first placing the objects that participate in

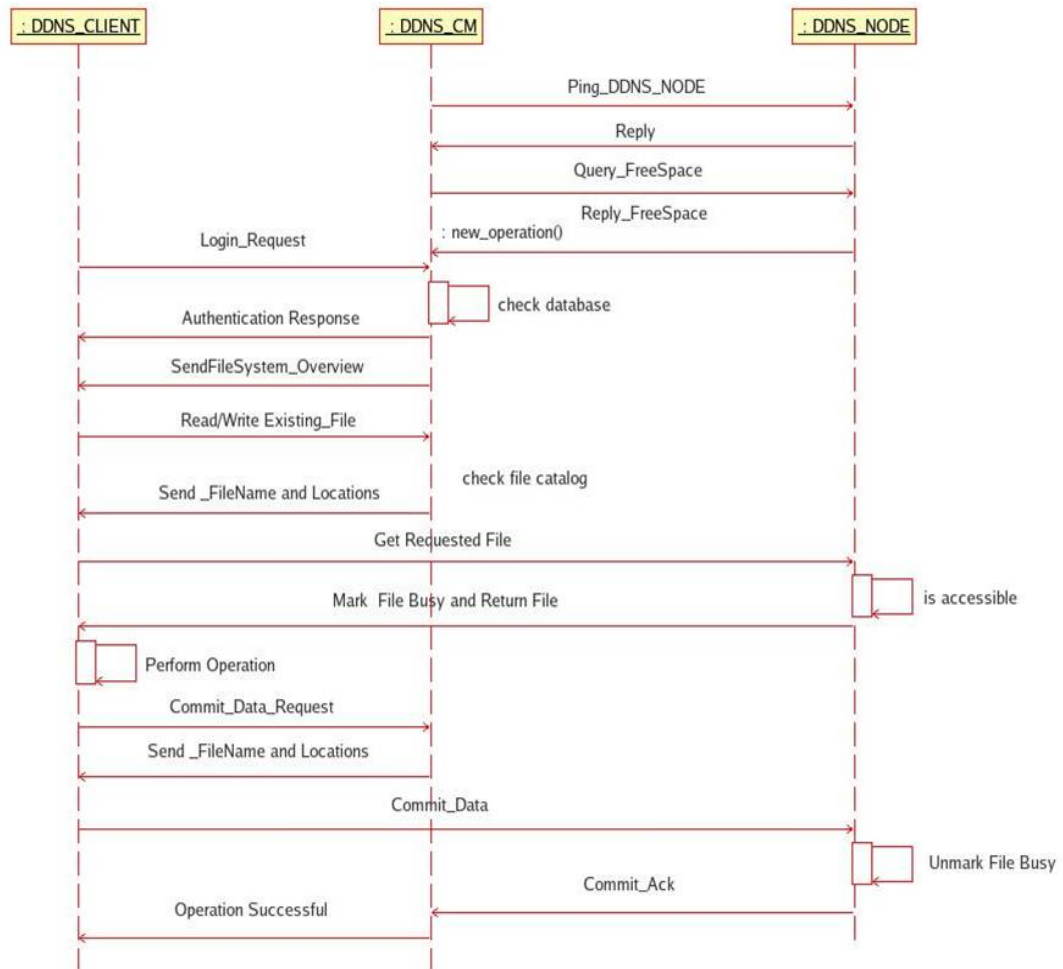


Figure 3.3: Sequence Diagram

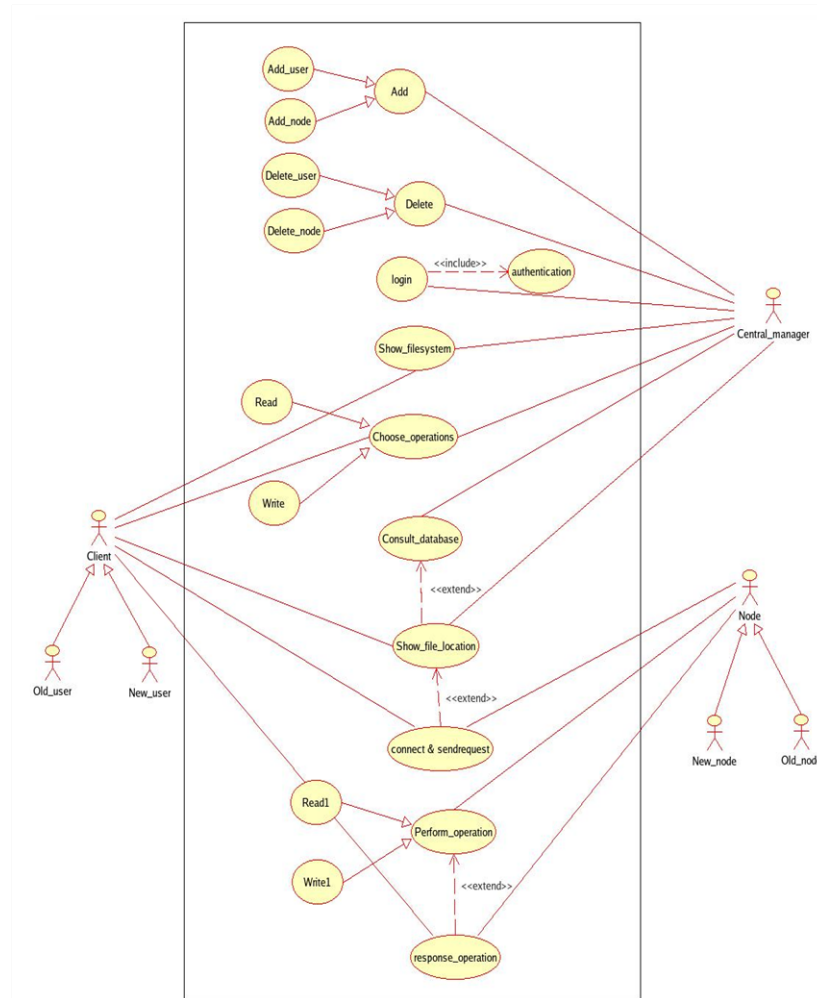


Figure 3.4: Use Case Diagram

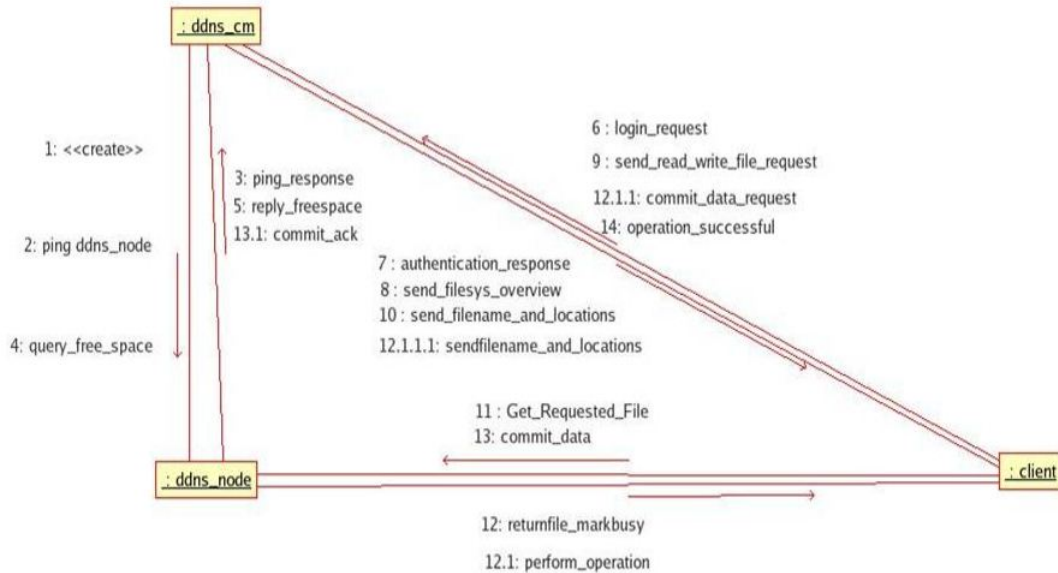


Figure 3.5: Collaboration Diagram

interaction as vertices in graphs. Next, you render the links that connect these objects as arcs of these graphs. Finally, you adorn these links with messages that objects send and receive. This gives the reader a clear visual cue to the flow of control in context of the structural organization of objects that collaborate. There are 2 features of collaboration diagrams

1. There is a path that indicates how one object is linked to another. You can attach a path stereotype to the far end of the link.
2. There is a sequence no. which indicates the time order of a message. We can prefix the message with a number.

3.2.5 Activity Diagram

Activity diagrams are one of the five diagrams in UML for modeling dynamic aspects of the system. An activity diagram 3.6 is essentially a flow chart, showing the flow of control from activity to activity. To use an activity diagram to model dynamic aspects of systems, this involves modeling the sequential

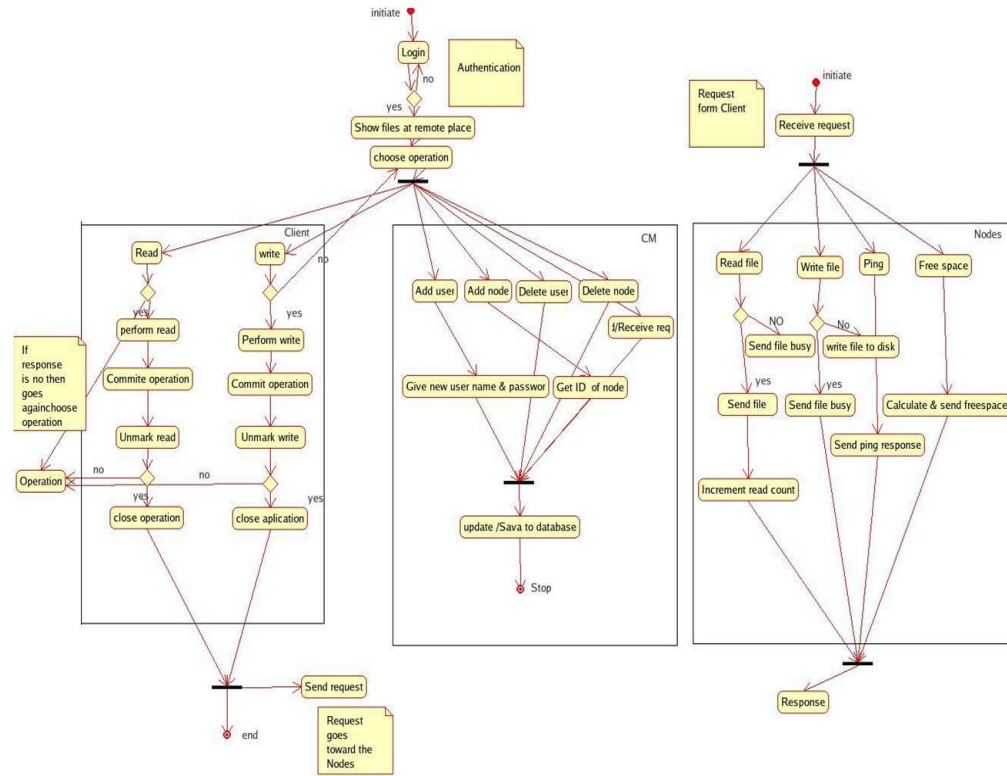


Figure 3.6: Activity Diagram

steps in computational process. With an activity diagram, you can also model the flow of object as it moves from state to state at different points in flow of control. Activity diagrams may also stand alone to visualize, specify, construct & document the dynamics of society of objects.

Activity diagrams are not only important for modeling the dynamics aspects of the system but also for constructing executable system through forward & reverse engineering.

3.2.6 Deployment Diagram

Deployment diagrams are one of 2 kinds of diagram used in modeling the physical aspects of an object oriented system. It shows the configuration of runtime processing nodes & components that live on them. We can use the deployment diagram to model the static deployment view of a system 3.7. This involves modeling the topology of hardware on which your system executes.

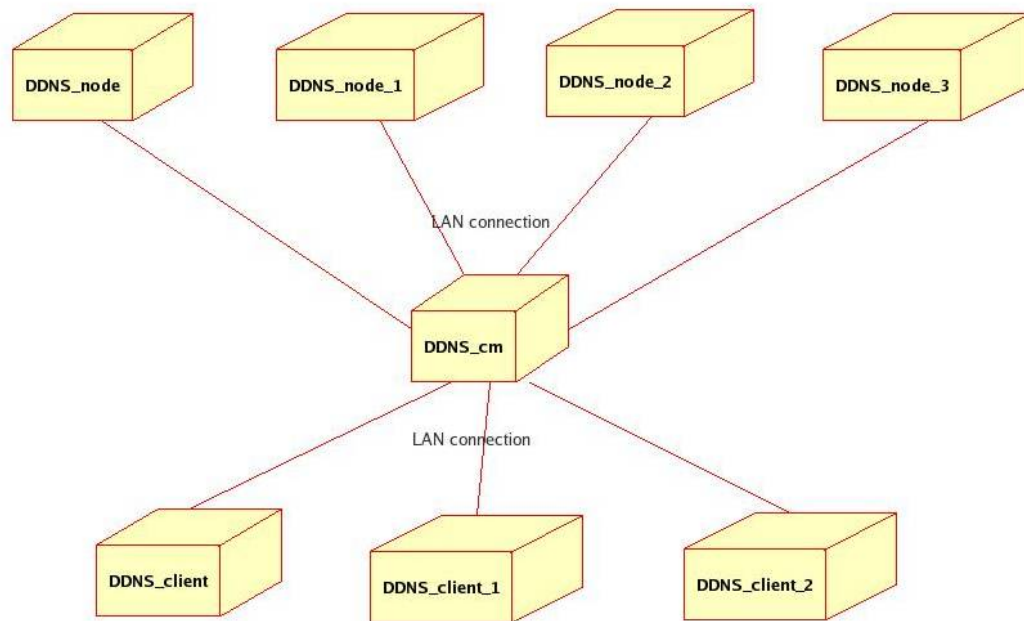


Figure 3.7: Deployment Diagram

Deployment diagrams are essentially class diagrams that focus on systems nodes.

Deployment diagrams are not only important for visualizing, specifying, & documenting embedded, clients / server, & distributed systems, but also for managing executable systems through forward & reverse engineering.

Chapter 4

Implementation

4.1 Remote Method Invocation

The basic structure of an RMI-based method call involves a client, a server and a registry. To make a call to a remote object, the client first looks up the object it wishes to invoke a method on in the registry. The registry returns a reference to the object (assuming it exists) on the server, which the client can use to invoke any methods that the remote object implements. The client communicates with the remote object via a User-defined interface that is actually implemented by the remote object. The client actually does not deal directly with the remote object at all, but with a code stub that deals with the process of communication between client and server (using, in the default case, sockets).

At the server end, in pre-Java 2 JDKs (before JDK 1.2), a code skeleton dealt with client communication. In Java 2, the skeleton is no longer needed. Fortunately, the code stub (and skeleton if necessary) are generated automatically by the RMI Compiler `rmic`. Remote objects can be invoked with parameters, naturally - these parameters can be entire objects, complete with methods that are passed using serialization. The basic architecture as shown in 4.1

The RMI server must implement the remote object's interface, and hence it must implement the methods in that interface. (It can in fact implement other methods as well, but such additional methods will only be available to other objects on the server to call - only those methods declared in the interface will be accessible remotely.) In addition, it is commonly the case that

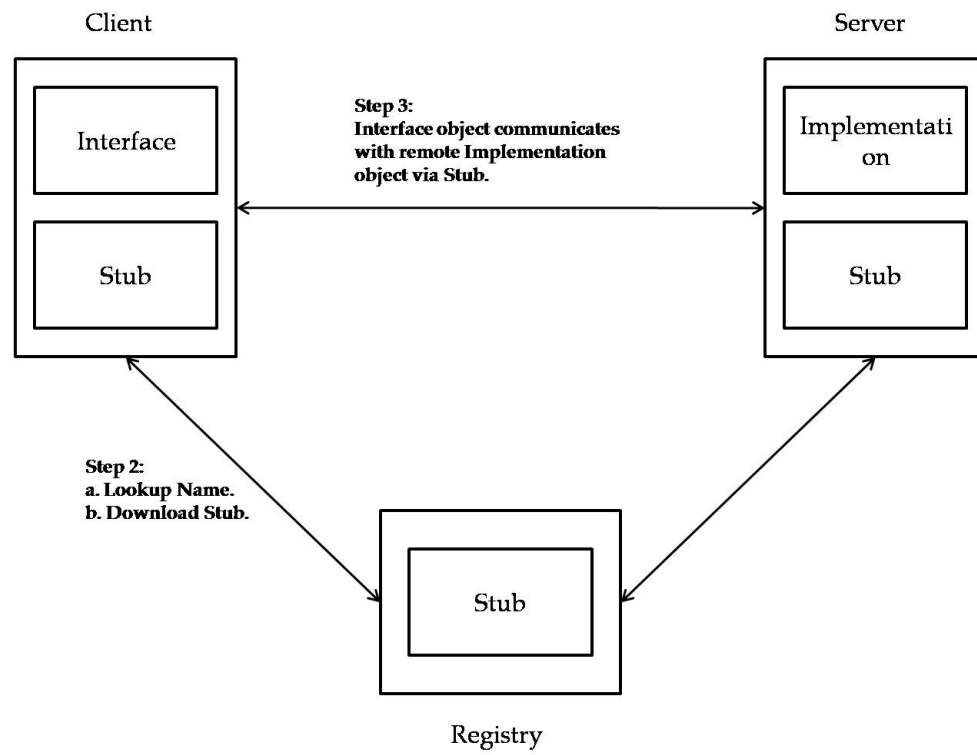


Figure 4.1: RMI Basic Architecture

remote objects extend the class `java.rmi.server.UnicastRemoteObject`, which provides the default behaviour required for RMI-based communication in 'typical' cases. It is also possible for you to define your own such behaviour, but that is beyond the scope of this module. In addition to the methods of the interface, the server must also undertake a certain amount of initialisation - make the remote object(s) available by binding them into the registry.

The client should set up a security manager (because the code stub(s) will be downloaded from the server and will then be checked), and must lookup the name of the remote object in the registry. The lookup process will return a reference to the remote object (strictly, to the code stub, but from the client's point of view it looks like the remote object). The remote object can then be treated exactly as if it was local - i.e. you can invoke methods on it in the same way. This is really the strength (and the point) of RMI: it allows distributed object-based programming using the same syntax and structures as local object-based programming with very little effort on the client side (or the server side come to that). We will now look at the stages of RMI in more detail.

4.1.1 Different Ways To Do It

There are a number of ways to set up and get RMI working. Some dependent on which JDK you are using. These are really just variations on the basic version.

4.1.1.1 RMI Interface

We need an interface describing a potential remote server object because the client code does not have a copy of the server code, but nevertheless needs to manipulate it. By defining an interface, we can make the server object's 'signature' available to the client.

The interface needed for constructing a remote server using RMI is the same as any other interface, except that it must extend the `Remote` interface and all the methods in it must declare that they can throw `RemoteException`. `RemoteException` is an exception that can occur when a failure occurs in the RMI process. By saying a method throws this exception we are saying that the method may generate that exception, but will not catch and handle it. It is then up to any methods that call the methods in the interface to catch this exception (or alternatively declare that they also throw it - which means it must be caught and dealt with elsewhere).

```
import java.rmi.*;
```

```

public interface FileServer extends Remote
{
    boolean writeFile(String fileName, byte[] buffer, int bufferStart, int bufferLength) throws Re-
moteException;
    byte[] readFile(String fileName) throws RemoteException;
    boolean mkdirs(java.io.File file) throws RemoteException;
}

```

Note that we have imported the classes `java.rmi.Remote` (the `Remote` interface).

4.1.1.2 RMI Server Implementation

The next step is to produce an implementation of the interface that will run on the remote server. The implementation will (in 'simple' cases) extend the class `UnicastRemoteObject`, which provides some necessary services that we would otherwise have to implement ourselves, and implements our remote interface.

```

import java.rmi.*;
import java.rmi.server.*;
import java.io.*;
public class FileServerImpl extends UnicastRemoteObject implements FileServer, NodeAdmin
{
    private String fileName;
    private int disks;
    public FileServerImpl() throws RemoteException
    {
    }
    public boolean writeFile(String fileName, byte[] buffer, int bufferStart, int bufferLength) throws
RemoteException {
        File tofile;
        BufferedOutputStream bos = null;
        try {
            tofile = new File(fileName);

```

```

    bos = new BufferedOutputStream(new FileOutputStream(tofile));
    bos.write(buffer, bufferStart, bufferLength);
    System.out.println(" Writting of " + fileName + " has started");
    bos.close();
    bos = null;
    System.out.println(" Writting of " + fileName + " finished");
}
catch (IOException ex)
{
    ex.printStackTrace();
    return false;
}
return true;
}

```

The Exception we catch is `RemoteException`: this catches all the communication-related errors that may occur. It has a large number of specific subclasses (for example, `UnknownHostException`) which can be dealt with separately if required.

4.1.1.3 RMI Clients

We now look at the client code which must find the object(s) it is looking for in the registry by looking up their names, and link it to a local object (actually it links to the code stub but we don't see that) for e.g.

```

import java.rmi.*;
public class EUStatsClient {
    public static void main (String args[]) {
        EUStats serverObject; Remote RemoteObject;
        if (args.length < 2)
        {
            System.out.println("usage: java EUStatsClient IP countryname");
            return;
        }
    }
}

```

```

    }
    System.setSecurityManager(new RMISecurityManager());
    try {
        String name = "rmi://" + args[0] + "/EUSTATS-SERVER";
        RemoteObject = Naming.lookup(name);
        serverObject = (EUStats)RemoteObject;
        System.out.println("Main language(s) of " + args[1] + " is/are " + serverObject.getMainLanguages(args[1]));
        System.out.println("Population of " + args[1] + " is " + serverObject.getPopulation(args[1]));
        System.out.println("Capital of " + args[1] + " is " + serverObject.getCapitalName(args[1]));
    }
    catch (Exception e)
    {
        System.out.println("Error in invoking object method " + e.toString() + e.getMessage()); e.printStackTrace();
    }
}
}
}

```

4.1.1.4 Compiling and Running

Next we need to compile and run the code. To make life easier, here is the interface, the server, the client and the policy file to download.

First, we consider the server. In the case of the server, we need to use the Java compiler to compile the RMI interface and the server code. We then need to use the RMI compiler to generate the stub code that the client will use [locally] to access the [remote] methods. The RMI compiler also generates the corresponding server-side 'receiver' objects, or skeletons, for dealing with communications from the client in pre-Java 2 code. Incidentally, the RMI compiler works by first generating source code for the stubs and skeletons, compiling it with `javac` and finally deleting the [intermediate] source code files: if you want to see what the code looks like, then use the `-keep` option to stop them being deleted. (Note that by default `rmic` generates code that is a little more complicated).

To compile the client code, we just need to do:

```
javac App.java
```

4.2 Advantages of RMI

At the most basic level, RMI is Java's remote procedure call (RPC) mechanism. RMI has several advantages over traditional RPC systems because it is part of Java's object oriented approach. Traditional RPC systems are language-neutral, and therefore are essentially least-common-denominator systems-they cannot provide functionality that is not available on all possible target platforms.

RMI is focused on Java, with connectivity to existing systems using native methods. This means RMI can take a natural, direct, and fully-powered approach to provide you with a distributed computing technology that lets you add Java functionality throughout your system in an incremental, yet seamless way.

The primary advantages of RMI are:

- **Object Oriented:** RMI can pass full objects as arguments and return values, not just pre-defined data types. This means that you can pass complex types, such as a standard Java hashtable object, as a single argument. In existing RPC systems you would have to have the client decompose such an object into primitive data types, ship those data types, and the recreate a hashtable on the server. RMI lets you ship objects directly across the wire with no extra client code.
- **Mobile Behavior:** RMI can move behavior (class implementations) from client to server and server to client. For example, you can define an interface for examining employee expense reports to see whether they conform to current company policy. When an expense report is created, an object that implements that interface can be fetched by the client from the server. When the policies change, the server will start returning a different implementation of that interface that uses the new policies. The constraints will therefore be checked on the client side-providing faster feedback to the user and less load on the server-without installing any new software on user's system. This gives you maximal flexibility, since changing policies requires you to write only one new Java class and install it once on the server host.

- **Design Patterns:** Passing objects lets you use the full power of object oriented technology in distributed computing, such as two- and three-tier systems. When you can pass behavior, you can use object oriented design patterns in your solutions. All object oriented design patterns rely upon different behaviors for their power; without passing complete objects-both implementations and type-the benefits provided by the design patterns movement are lost.
- **Safe and Secure:** RMI uses built-in Java security mechanisms that allow your system to be safe when users downloading implementations. RMI uses the security manager defined to protect systems from hostile applets to protect your systems and network from potentially hostile downloaded code. In severe cases, a server can refuse to download any implementations at all.
- **Easy to Write/Easy to Use:** RMI makes it simple to write remote Java servers and Java clients that access those servers. A remote interface is an actual Java interface. A server has roughly three lines of code to declare itself a server, and otherwise is like any other Java object. This simplicity makes it easy to write servers for full-scale distributed object systems quickly, and to rapidly bring up prototypes and early versions of software for testing and evaluation. And because RMI programs are easy to write they are also easy to maintain.
- **Connects to Existing/Legacy Systems:** RMI interacts with existing systems through Java's native method interface JNI. Using RMI and JNI you can write your client in Java and use your existing server implementation. When you use RMI/JNI to connect to existing servers you can rewrite any parts of you server in Java when you choose to, and get the full benefits of Java in the new code. Similarly, RMI interacts with existing relational databases using JDBC without modifying existing non-Java source that uses the databases.
- **Write Once, Run Anywhere:** RMI is part of Java's "Write Once, Run Anywhere" approach. Any RMI based system is 100% portable to any Java Virtual Machine*, as is an RMI/JDBC system. If you use RMI/JNI to interact with an existing system, the code written using JNI will compile and run with any Java virtual machine.
- **Distributed Garbage Collection:** RMI uses its distributed garbage collection feature to collect remote server objects that are no longer referenced by any clients in the network. Analogous

to garbage collection inside a Java Virtual Machine, distributed garbage collection lets you define server objects as needed, knowing that they will be removed when they no longer need to be accessible by clients.

- Parallel Computing: RMI is multi-threaded, allowing your servers to exploit Java threads for better concurrent processing of client requests.
- The Java Distributed Computing Solution: RMI is part of the core Java platform starting with JDK?? 1.1, so it exists on every 1.1 Java Virtual Machine. All RMI systems talk the same public protocol, so all Java systems can talk to each other directly, without any protocol translation overhead.

Chapter 5

Working with Distributed Data over Network System (DDNS)

5.1 Introduction

We have built 3 modules namely,

- **DDNS Central Manager (DDNS_CM)**,

DDNS Central Manager maintains the Database of File systems, information about users & information about Nodes.

- **DDNS Node (DDNS_NODE)**,

DDNS Node stores the data stored by Central Manager

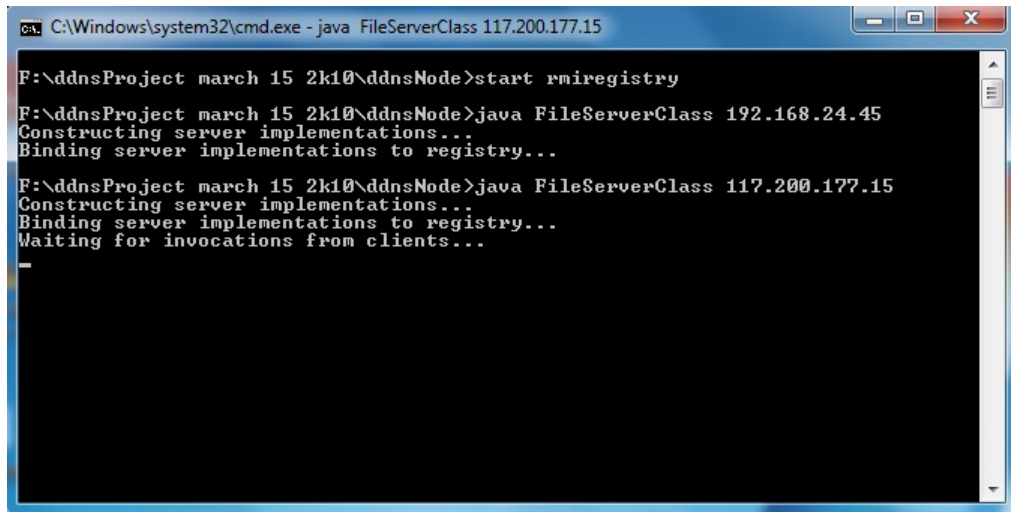
- **DDNS Client (DDNS_CLIENT)**.

DDNS Client opens the file in Read/Write mode.

5.2 DDNS_NODE

It is the place where storage of data will take place.

How to start DDNS_NODE:



```
C:\Windows\system32\cmd.exe - java FileServerClass 117.200.177.15

F:\ddnsProject march 15 2k10\ddnsNode>start rmiregistry

F:\ddnsProject march 15 2k10\ddnsNode>java FileServerClass 192.168.24.45
Constructing server implementations...
Binding server implementations to registry...

F:\ddnsProject march 15 2k10\ddnsNode>java FileServerClass 117.200.177.15
Constructing server implementations...
Binding server implementations to registry...
Waiting for invocations from clients...
_
```

Figure 5.1: Starting Node

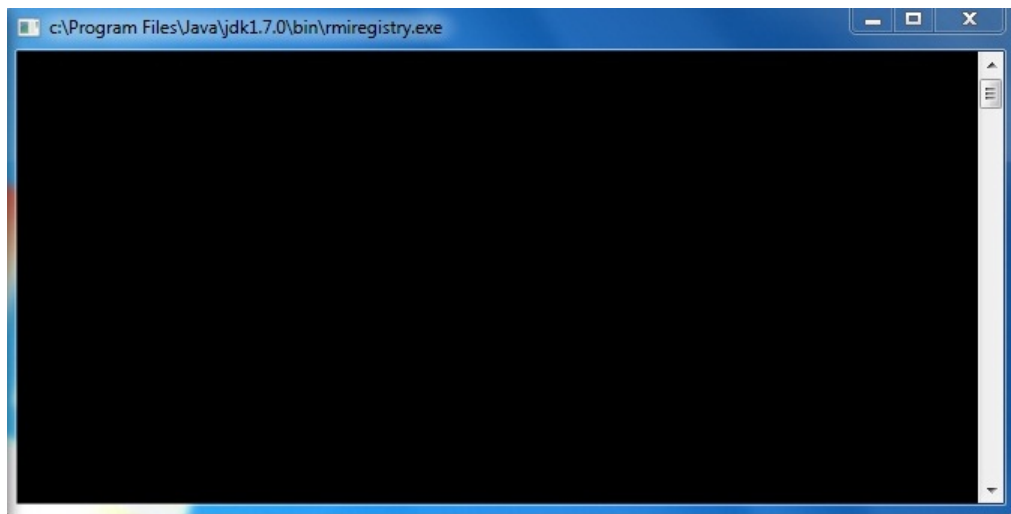
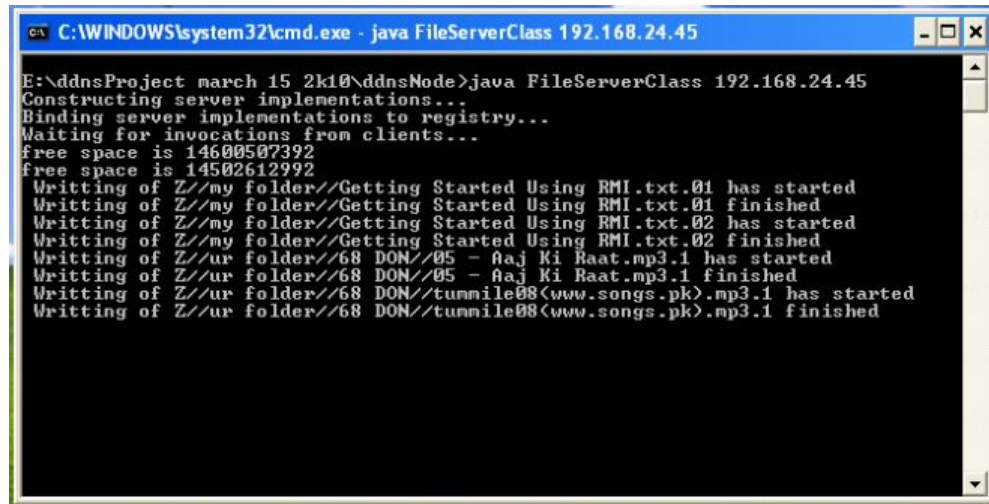


Figure 5.2: RMI Registry



```
C:\WINDOWS\system32\cmd.exe - java FileServerClass 192.168.24.45
E:\ddnsProject march 15 2k10\ddnsNode>java FileServerClass 192.168.24.45
Constructing server implementations...
Binding server implementations to registry...
Waiting for invocations from clients...
free space is 14600507392
free space is 14502612992
Writting of Z:/my folder//Getting Started Using RMI.txt.01 has started
Writting of Z:/my folder//Getting Started Using RMI.txt.01 finished
Writting of Z:/my folder//Getting Started Using RMI.txt.02 has started
Writting of Z:/my folder//Getting Started Using RMI.txt.02 finished
Writting of Z:/ur folder//68 DON//05 - Aaj Ki Raat.mp3.1 has started
Writting of Z:/ur folder//68 DON//05 - Aaj Ki Raat.mp3.1 finished
Writting of Z:/ur folder//68 DON//tunnile08<www.songs.pk>.mp3.1 has started
Writting of Z:/ur folder//68 DON//tunnile08<www.songs.pk>.mp3.1 finished
```

Figure 5.3: Directory creation and File Storing

1. For windows XP OS, give command on Windows XP: **start rmiregistry** at command prompt, new window will appear 5.2, which indicates successful starting of rmiregistry.
2. For Linux OS, give command **rmiregistry &** at terminal's prompt, this will start rmiregistry as background process 5.4.
3. Than give command **java FileServerClass ipaddress.** eg. java FileServerClass 192.168.24.45. 5.1.
4. If you see 5.1 and 5.2, it means node has started successfully.

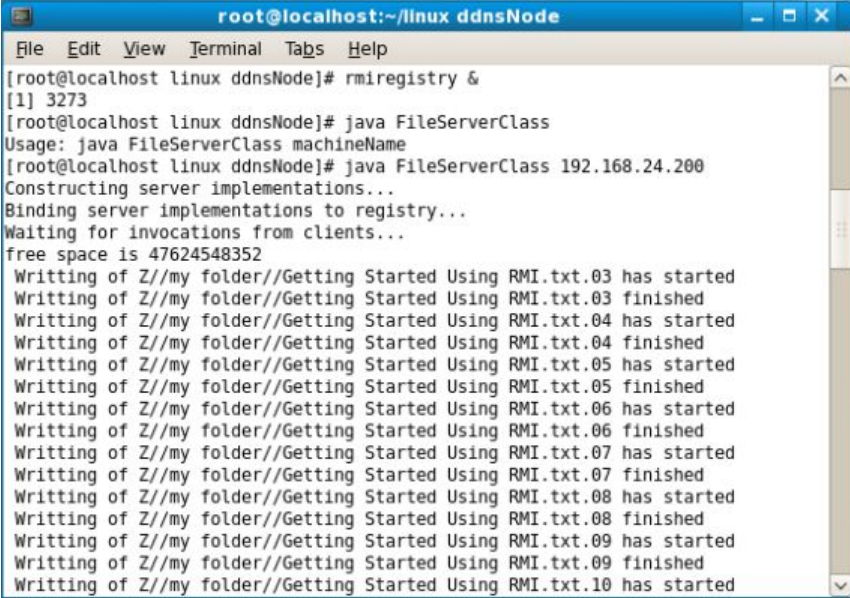
As per the Ddns Server Admin's requests the respective directories will be created and the computed number of splits will be stored in respective directories.

DDNS_NODE : Windows Node.

DDNS_NODE : Linux Node.

5.3 DDNS_CM

Central Manager should be ready with Oracle 9i Database having following tables:

A terminal window titled 'root@localhost:~/linux ddnsNode' with a menu bar (File, Edit, View, Terminal, Tabs, Help). The terminal shows the following commands and output:

```
[root@localhost linux ddnsNode]# rmiregistry &
[1] 3273
[root@localhost linux ddnsNode]# java FileServerClass
Usage: java FileServerClass machineName
[root@localhost linux ddnsNode]# java FileServerClass 192.168.24.200
Constructing server implementations...
Binding server implementations to registry...
Waiting for invocations from clients...
free space is 47624548352
Writing of Z//my folder//Getting Started Using RMI.txt.03 has started
Writing of Z//my folder//Getting Started Using RMI.txt.03 finished
Writing of Z//my folder//Getting Started Using RMI.txt.04 has started
Writing of Z//my folder//Getting Started Using RMI.txt.04 finished
Writing of Z//my folder//Getting Started Using RMI.txt.05 has started
Writing of Z//my folder//Getting Started Using RMI.txt.05 finished
Writing of Z//my folder//Getting Started Using RMI.txt.06 has started
Writing of Z//my folder//Getting Started Using RMI.txt.06 finished
Writing of Z//my folder//Getting Started Using RMI.txt.07 has started
Writing of Z//my folder//Getting Started Using RMI.txt.07 finished
Writing of Z//my folder//Getting Started Using RMI.txt.08 has started
Writing of Z//my folder//Getting Started Using RMI.txt.08 finished
Writing of Z//my folder//Getting Started Using RMI.txt.09 has started
Writing of Z//my folder//Getting Started Using RMI.txt.09 finished
Writing of Z//my folder//Getting Started Using RMI.txt.10 has started
```

Figure 5.4: Directory creation and File Storing on Linux Node

1. users.
2. node.
3. file_sys.

CHILD_SET is a varray type in oracle as:

create type child_set as varray(400) of int;

number 400 indicates the number of files/folders inside a folder, that can be handled safely by this DDNS_CM.

Tables are as follows:

We have 2 sub modules in DDNS_CM as:

- **DDNS Server Admin,**

DDNS Server Admin does the work of administration.

- **DDNS Server.**

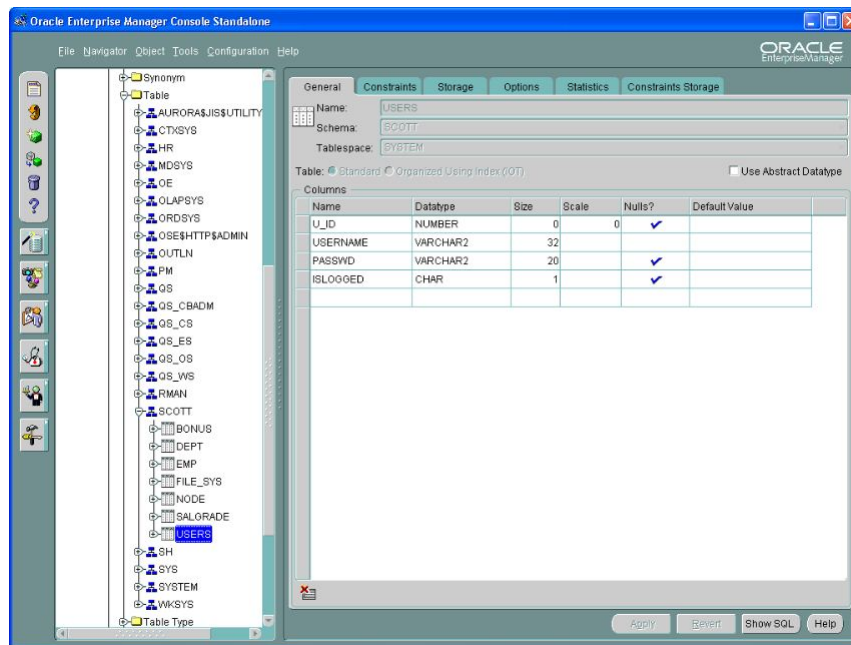


Figure 5.5: Table : Users

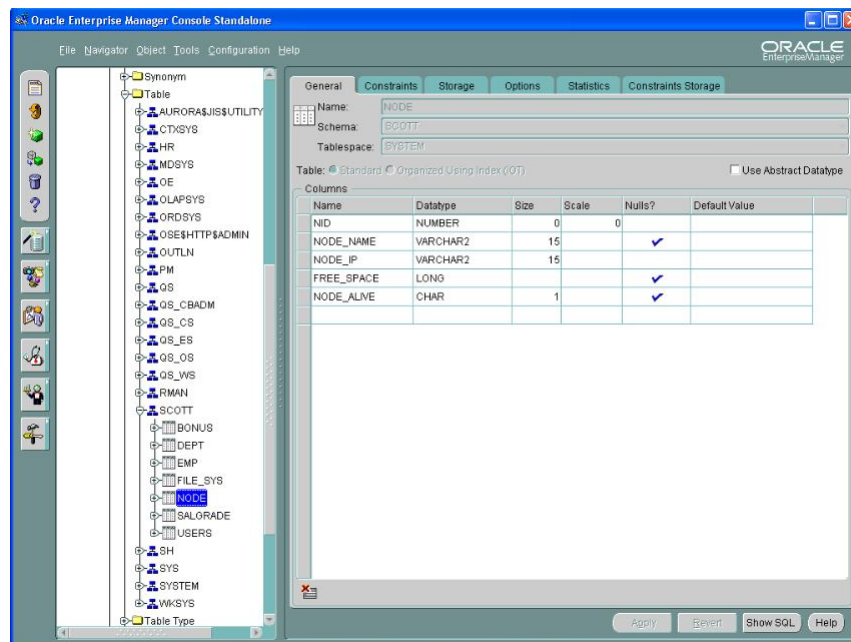


Figure 5.6: Table : Node

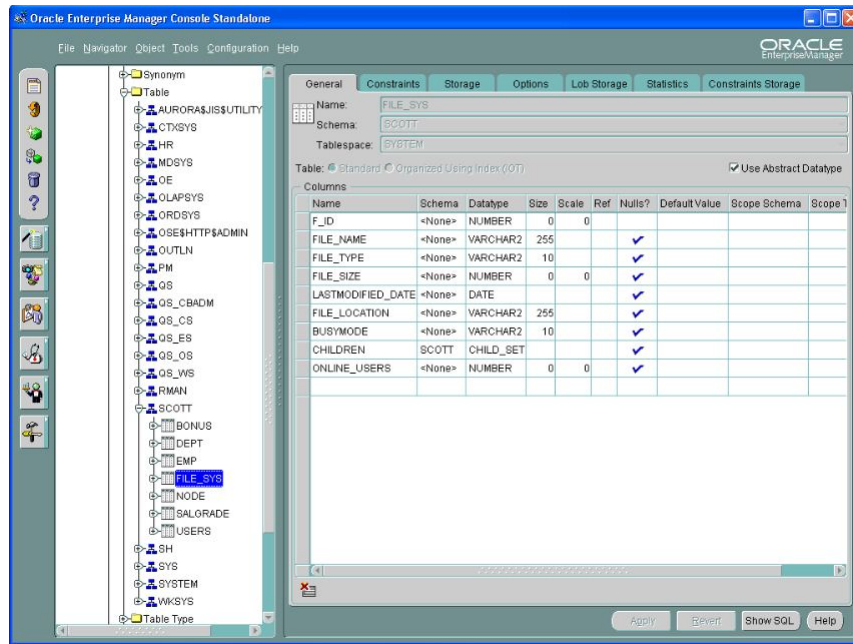


Figure 5.7: Table : File system

DDNS Server provides the service to the client.

5.3.1 DDNS Server Admin

This sub-module allows the administrator to administration tasks, such as Add or Remove User, Add Node and Storing Files on various nodes.

5.8 is the Login Page for administrator.

After successfully login, administrator will see: 5.9

As mentioned above administration consists of 5.10:

1. Add or Remove User.
2. Add Node.

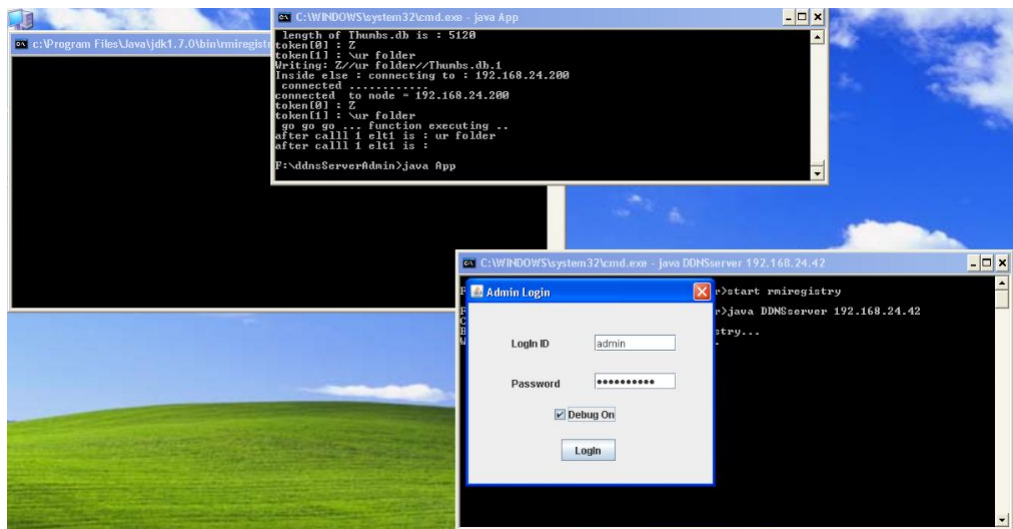


Figure 5.8: Admin Login

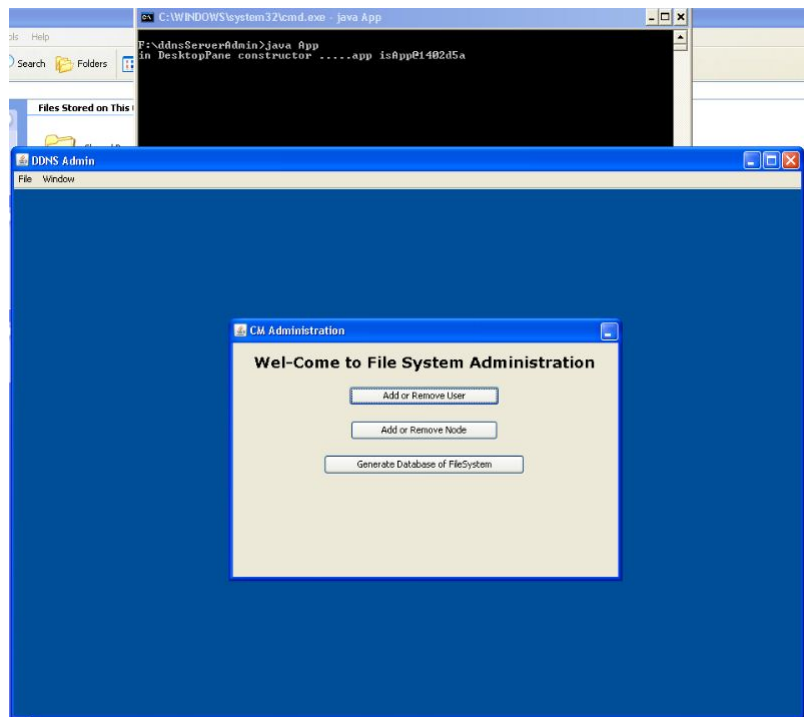


Figure 5.9: Main Screen

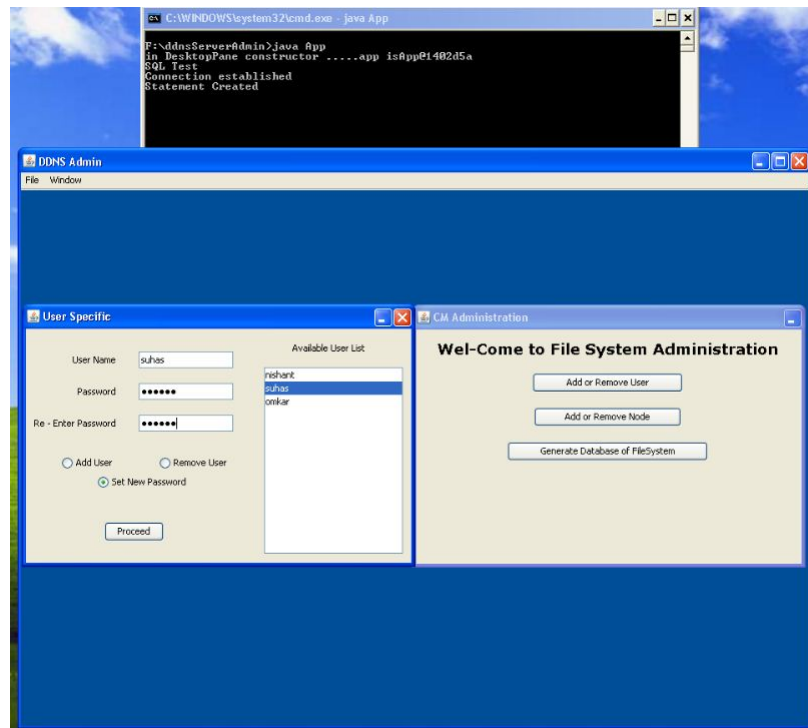


Figure 5.10: Add or Remove User

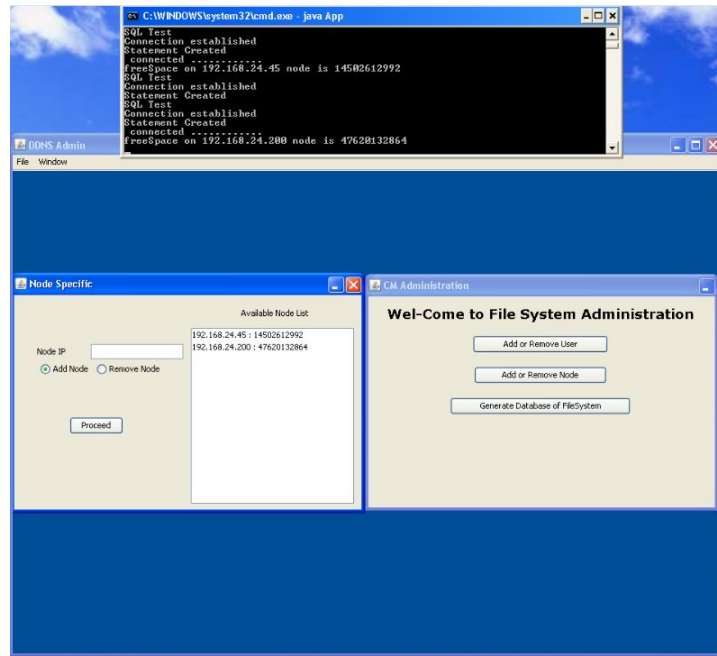


Figure 5.11: Add Node

3. Generate Database of FileSystem.

Here administrator can Add or Remove the User by referring the Users list shown aside, And he can change the password for any user, if necessary.

Both the username and password are stored in the database in encrypted form. They are encrypted using PBE (Password Based Encryption) technique.

Here available node list is displayed aside, administrator can add new node 5.11.

When administrator clicks on Proceed button, Connection is established to the specified node and available free space on the parent drive is returned by that node, provided that DDNS_NODE 5.1 is running in that node.

This free space is stored in the database.

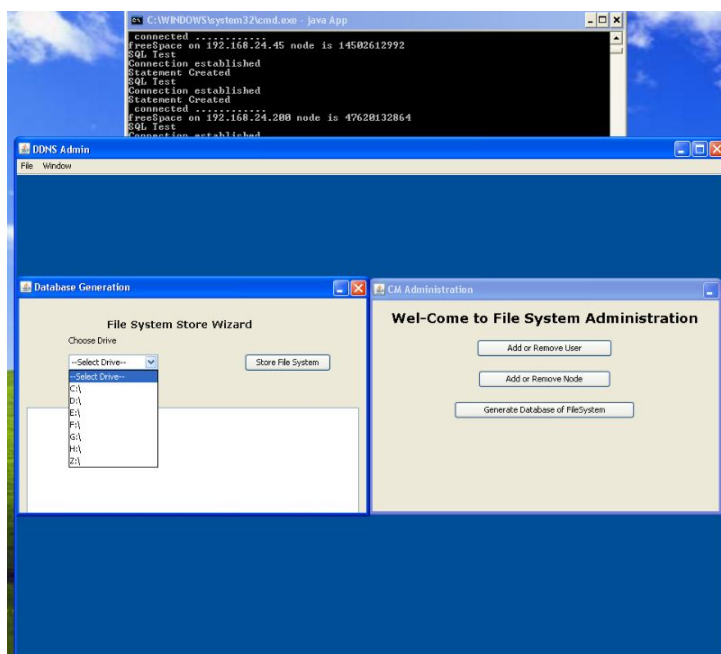


Figure 5.12: Choose drive

Caution : before clicking on Generate Database of FileSystem, administrator must have added nodes.

Administrator needs to choose the drive, so as to distribute and store the files on DDNS_NODE 5.13.

As shown above files on Z:\ drive are distributed and stored on the nodes 5.11.

After successful distribution and storing of files on nodes, initial administration is nearly finished.

5.3.2 DDNS Server

Caution : before starting DDNS Server, initial administration must have finished related to the generating database entries and storing of file system 5.13

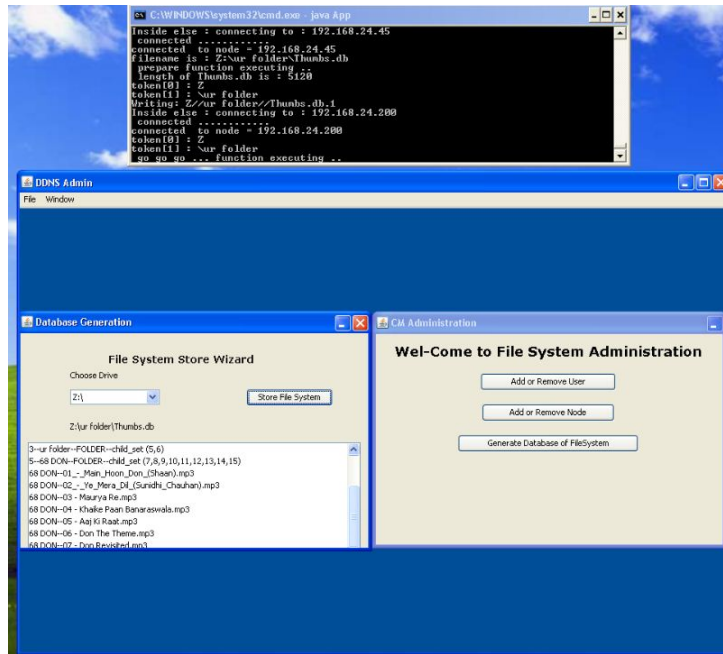


Figure 5.13: Store File System

How to run DDNS Server:

1. In windows xp, give command **start rmiregistry** at command prompt. New window will appear
2. Give command **java DDNSserver ipaddress** at command prompt, where ipaddress is IP Address of DDNS_CM.

Above figure 5.14shows successful starting of DDNS Server.

5.4 DDNS_CLIENT

Caution : before starting client DDNS_NODE and DDNS_CM must be running at respective machines.

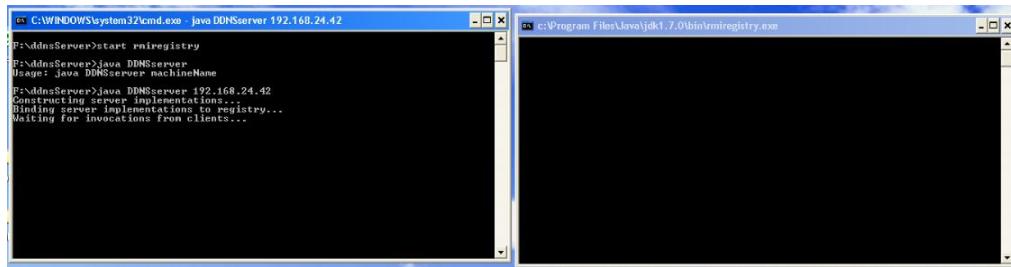


Figure 5.14: Starting DDNS Server



Figure 5.15: Client Login

Note: Files are opened in windows application and even in DDNS_CLIENT application in editor provided by this project.

How to run DDNS Client:

1. In windows xp, give command **java App serverIP** at command prompt.
2. Client Login Window appears as 5.15.

After successful login of the user, the user will see the File system hierarchy as 5.16.

When a File is selected in Read / Write mode a thread is created as 5.17.

After opening a File in Read mode, parts of that file are fetched at the client side for reading purpose and a connection successful window appears after merging all the parts at client side as

Now the File Getting started using RMI is opened in Read mode for reading as

These are some Client GUI features as 5.20, 5.21:

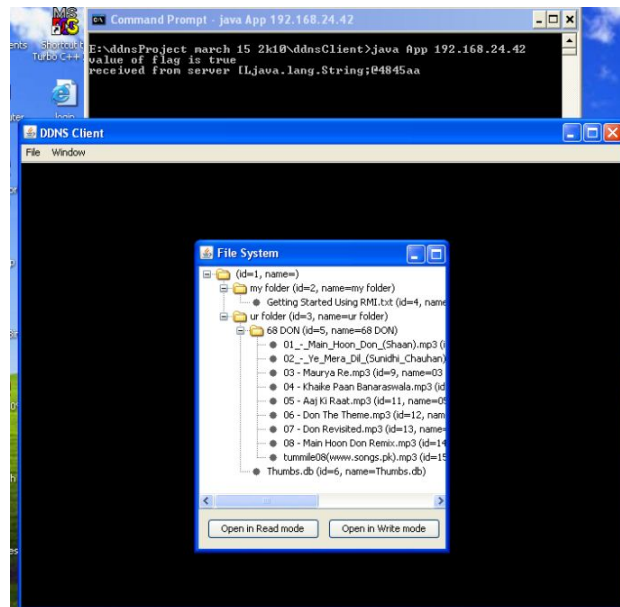


Figure 5.16: File System Hierarchy

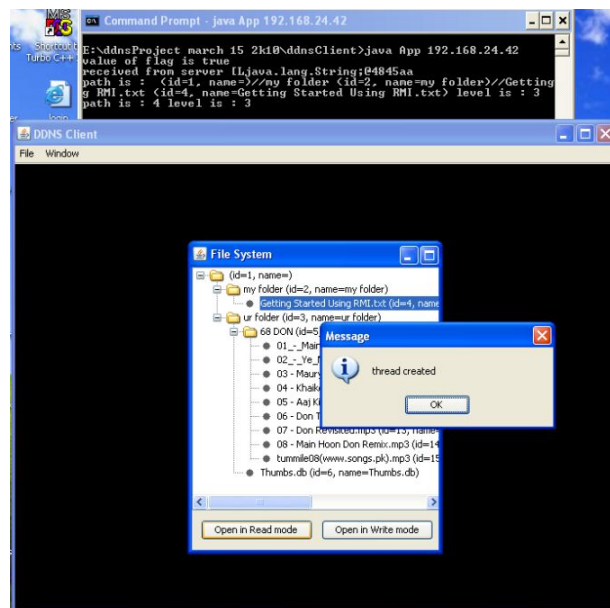


Figure 5.17: Thread Created

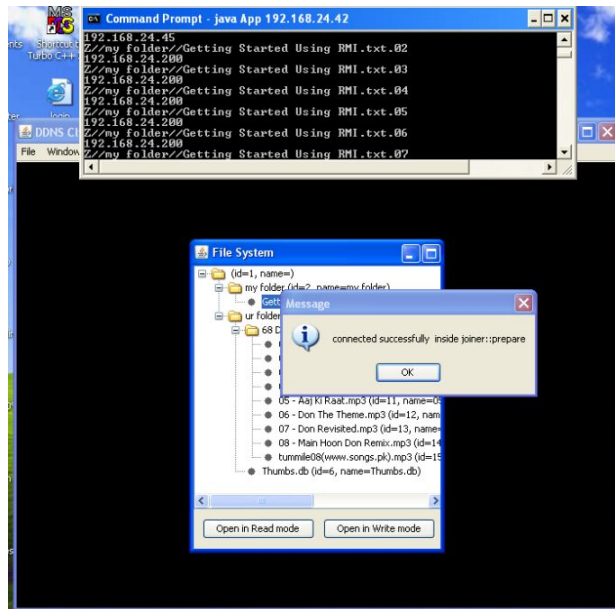


Figure 5.18: connection successful

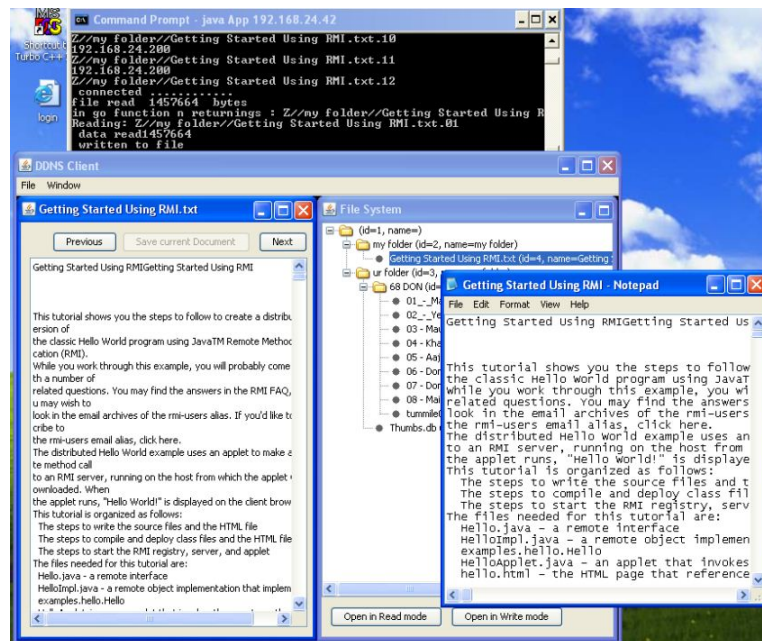


Figure 5.19: Getting started using RMI

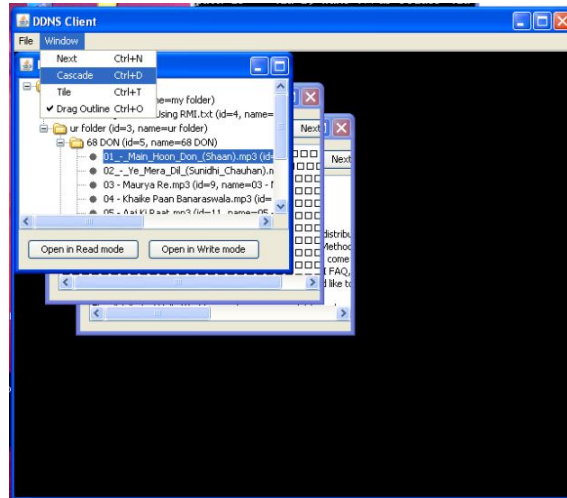


Figure 5.20: Cascade

- Cascade Windows.
- Tile Windows.
- Drag OutLine.

Whenever requested file is busy, client will see (observer the second last line of command prompt) 5.22:

User is trying to open the file “Getting started using RMI.txt” in write mode, but the file is opened by other user in read mode, hence user requesting file opening in write mode will be informed to try later.

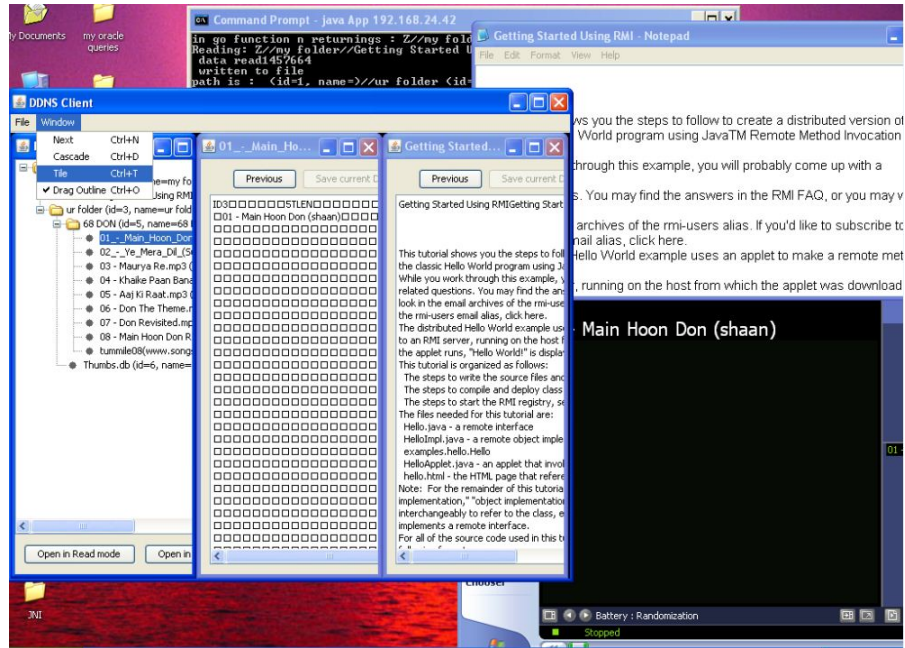


Figure 5.21: Tile

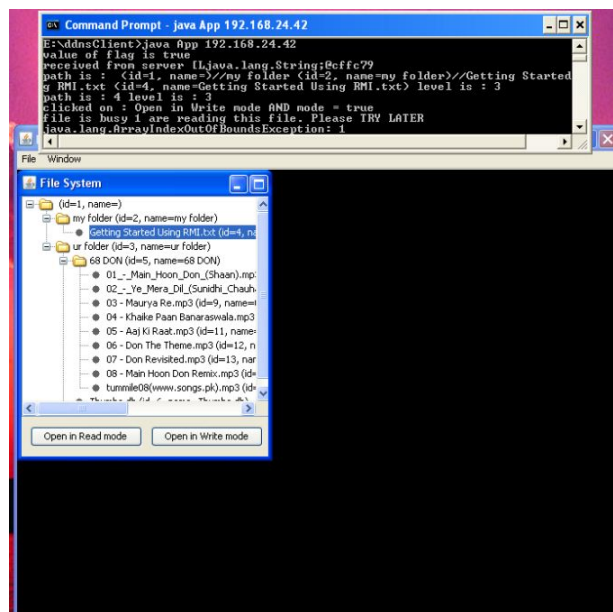


Figure 5.22: File is Busy

Chapter 6

Conclusion

Distributed Data over Network system allows a group of computers (nodes) to be combined into a single network with the Central Manager monitoring over them. The system allows the mass storage over the nodes in LAN.

The files are stored on the selected nodes' dedicated drives maintaining the actual file system hierarchy. By considering the free space available on that dedicated drive as investment of that node. Similarly, every node with such available free space on their dedicated drives is their investment for storing the files. Summing up the total free space on all nodes gives total investment by all nodes. Number of splits of file are stored on the nodes depending on their free space as their share. So the distribution of files on nodes is done uniformly. And every file is splitted and computed number of splits are stored on nodes. So the client is having access to the file without having any space overhead at client side, as the required split is only requested. So the low space problem at client side doesnt affect the client while accessing the files from server.

Thus effective storing of files is done on the nodes for the ease of the client.

Bibliography

- [1] Google File System.
- [2] Parallel Virtual File System.
- [3] www.clemson.edu.