Java Distributed File System

Tomasz Mista

MSc Advanced Computer Science
Session 2010/2011

MSc Project - COMP5200
Student #200592050

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.
I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student)……

Abstract

MSc in Advanced Computer Science 2011

Java Distributed File System

By Tomasz Mista

Project Supervisor: Haiko Muller

There are many ready implemented distributed file systems for businesses and individuals. However, some of the problems that users faces when using those systems are still challenging for scientist. The ideal distributed system would provide shared access to the same set of files to all its users, would be scalable to provide more storage space and higher performance to a growing user community. It would require minimal human administration, and it would not become more complex as more components were added.

This dissertation describes my own Java Distributed File System (JDFS) as a new challenge in solving problems occurring in currents file distributed systems. It defines the concepts of transparency, scalability and fault tolerance and solves them in JDFS approach. It also presents alternatives for the semantics of sharing and methods for providing access to remote files. A survey along different distributed systems for file sharing demonstrates various implementations and design strategies. Based on the assessment of those systems and own research new approaches are provided in implemented system.

The paper ends with a critical examination of the product and discusses possible improvements which could be made in the future.

Acknowledgements

I would like to thank my supervisor Dr. Haiko Muller for his support and the motivation he gave me through the project.

Contents

List of Figures

List of Tables

Chapter 1: Introduction

Nowadays, new applications require more resources than are available on an inexpensive machine. The organisations face problems with business processes that no longer fit on single cost effective machine. Also individuals have a certain amount of data stored on their home or business computers that is sensitive. Loss of this data would stop organizations to run properly or would incur a significant cost in terms of financial cost or time and effort. Sensitive data on a personal home computer could be a student's university assignment. If this assignment is lost, it will incur a time cost to the student to redo the assignment. The Distributed File System (DFS) was invited to help business and individuals run their processes more effectively and without data lost during system failure or crash.

Who doesn't know about Google, Facebook, Youtube and other services whose allows users to upload pictures, videos in their servers. Now Google is beyond the searching because it supports uploading videos, it gives email account of few gigabytes, it has Google Map, Google Earth and Google News application. All these application are heavily data intensive however Google provides their services very efficiently. The question is how Google and other companies do it? Let's look on the figure below (**Figure 1**). Yes, that's right they use DFS to provide their services on very high level.

Figure 1: DFS is now everywhere and most companies use DFS to provide various services to their users.

This paper addresses the technical challenges of creating JDFS (Java Distributed File System) as an environment for sharing files between different groups of users. In this JDFS attention to details must be paid such as transmission and security access of shared files. There are two main preconditions of a great design for JDFS, and there are as follow:

- Be familiar with features of current DFS and know the relevant aspects of peer-to-peer and server/client based architecture.
- Know the characteristic of java-network-oriented and object-oriented characteristic because Java is the programming language for this system.

1.1 Structure

In light of these objectives, the thesis will take the following structure:

- A literature review about Distributed File Systems, more detailed aspects of DFS will be provided and explained such as: synchronous and asynchronous message passing, protocol, file replication (Chapter 2). Chapter 3 defines the methodology chosen in order to deliver valuable project. DFS analysis and evaluation of current DFS will be given in Chapter 4: Current System Analysis where detailed comparison is included. Chapter 5: Initiation Project shows feasibility analysis to make sure the project will success. Chapter 6: Java DFS shows Java object-oriented and network-oriented techniques and powerful platform, which indicates why the Java is the best selection for the development of JDFS.
- The second section of document focus on the actual system analysis and design where UML diagrams are drawn. Chapter 10 Implementation, it is to detail the implementation of the system in order to make the system work flows more accurate and fluent
- The final part of this report will be the system testing and final project conclusion. System testing has to be done in Chapter 11, which makes sure that the JDFS can be accepted by the user and potential market.

1.2 The problem

A distributed file systems have grown in importance in recent years because as a reliance increase, the problem of availability becomes more acute. Today, individuals and business alike have a certain amount of data stored on their home or business computers or both. Those data are essential for the day-to-day work or activities of users. But, this work can be disrupted by several problems:

- Inability to upload any file in any size because unsupported file type or extended file size.
- Inability to create file sharing permissions for example some files we want to share only with friends and other with groups or publish to the world.
- Inability to restrict permissions on a file by a single user (read, write, delete, rename)
- Lack of fault tolerance because one service is down, the user cannot access another, either manually or automatically.
- Transporting many files over the intranet can create slow performance and network bottleneck.
- Inability to upload a file because incompatible file format
- The security of data is an important issue because we have to make sure that only authorized users have access to information.

All those problems can cause serious disruptions in a daily user's work of individuals or businesses, and a good distributed system must deal with all those problems in proper way.

1.3 Aims and objectives

The aim of this project is to build a Java Distributed File System (JDFS) providing secure files storage, location independence, scalability and transparent migration capabilities for data.

Objectives in order of priority:

1. All users are able to upload and download a file to and from the system.
2. All users are given a consistent view of the same set of files.
3. All users can specify file sharing permissions: read, write, delete, rename.

4. Investigate different techniques for avoiding problems such as: bottleneck, slow performance, security, bad file replications.
5. Produce a working prototype incorporating techniques specified above and show how they can be integrated with distributed file system.
6. Evaluate the project process on the whole.

Future enhancements: The project can be extended by deeper research into the concept of distributed file system and new area of use for such system such as mobile computing. Future advanced techniques and algorithms can be analyzed and used for more effective file replications.

1.4 Proposed Approach

The main approach is to design and develop a JDFS (Java Distributed File System) that meets the user's requirements and specification. Before deciding on the types of approach, there are number of factors that need to be considered:

- **Technology** – this project will involve software components. Most of the components are easily available in the market. Therefore, searching for the best components will be carried out.
- **Cost** – the budget of building prototype should be kept as low as possible
- **Product requirements** – the JDFS should be easy to use according to user's needs. Requirements and specifications will be obtained from external sources and as well as from the use of literature review.

Besides, I have to ensure that building this project will not have any similarities or duplications of current distributed file systems used.

The following methods will be employed in this project:

- Specification definition
- Software design and development
- Implementation
- Integration and testing

1.5 Deliverables

The project deliverables are as follow:

- Report with analysis of current distributed files systems: Sun NFS and Hadoop, analysis, design and implementations steps of JDFS with proofing that Java is a reasonable choose for implementations of various distributed systems
- System prototype with functionalities stated in project requirements, system testing results in order to state future system improvements.

1.6 Required Resources

The resources I will need are accessible from School of Computing. The programming environment needed (Eclipse) is Open Source, the prototype will run on several lab machines (during the testing and presentation to assessor) those are available, so it is therefore not a formal resource requirement.

1.7 Scope of project

Others goals of the project are to master my skills in software design and programming because the project takes the challenge to develop a complex system for distributing files over the Intranet. The challenge is to perform a deep research into literature and current DFS's. It is necessary because I am going to build this system mainly from scratch to make sure that the system will have no similarities

in market. The codes written must be in 100% unique, so the project success mainly will depend on my programming skills. I want to make sure that the system is using interesting features and techniques for file distribution.

Also, the project includes UML diagrams drawing and the programming includes the learning of advanced techniques from networking. Most of this skills used in this project were not touched upon in regular classes are taken here.

1.8 Time Management

As with any project, time management is of crucial importance for the purpose of generating more effective work and productivity. Time is finite. We have only so many hours to do what we need to do and what we want to do [10].

1.8.1 Project Plan

For this project I used GanttProject software to plan out activity over the course of 12 weeks allocated.

The chart attached (**Appendix 1**) shows an example plan. It is a screenshot of the plan as it was during the literature review. The plan changed many times over the course of the project, but its use was invaluable in managing the scope of the work.

There are some aspects missing from the plan at this stage, and also that there are things that have been planned. This is because the plan is a living document, and changes constantly.

In order to gain better understanding of the DFS and in technical skills and tool required to implement the project requirement an extensive review of literature on the relevant subject was carried out. The literature review for the project will include the following:

- Clarify the meaning of DFS, give a brief history, identify the main functions, and main area of operations for such system.
- Clarify the meaning of: protocol, file replication and namespace in order to understand how DFS works.

Future research related to DFS will be included in later chapters by providing more advanced research.

2.1 DFS Concept

Distributed File System (DFS) support the sharing of information in the form of files throughout an intranet. A well designed file service provides access to files stored at a server with performance and reliability similar to disk [4]. A DFS allows users to store and access remote files like in the local way, but from any computer in the intranet.

In other words a DFS system is capable of distributing files from one computer to other computers or storage devices, maybe one group to another group. This is just a personal sense from the first sight of distributed file system, which cannot make sense of it clearly. In scientific field, a DFS is a distributed implementation of the classical time-sharing model constructed on a file system, where many users can share files and storage resources.

Distributed File System supports effectively common operations on the same kind of sharing resource, especially for the files physically dispersed among the different sites of a distributed system. From the view point of inheritance, a DFS can be seen as a system inherited from Distributed System and File System. A Distributed System which centralized systems from a set of divers, autonomous, physically dispersed computer systems which are inter connected by communications networks to permit the exchange of information [9]. So a Distributed System is a resource set of processors that do not share memory or clock. It distributed each processor to each appropriate site by varied functions. Similarly with a File System, a DFS also highly provides common files operation, such as delete, read, write, rename, copy and share which formed client interface. Not only are these operations used in certain specifically local site of DFS, but also used in other sites through proper authorization access. But the operation creating a file, making file unshared and setting the permission of file are still constrained in local site. If a Distributed system and a File system are both compared as super classes, then a DFS can be seen as a polymorphic subclass which accommodates its resource (processors) distributed mechanism derived from Distributed System and files operation inherited from File System. The figure below (**Figure 2**) shows the relationship among Distributed System, File System and DFS.

The [4] states that DFS can be implemented as a part of a Distributed Operating System. A Distributed Operating System is nothing but the collection of computers which are interconnected and that appears to the user as a single system, and if any of the system crashed, then it does not affect the other system [4]. Alternatively speaking it can be seen as a bridge between conventional Operating System and File System (**Figure 2**). A distributed Operating System manages the data (such as file) and process migration from one site to another, so DFS play the same role with Distributed Operating System on controlling file migration between sites.

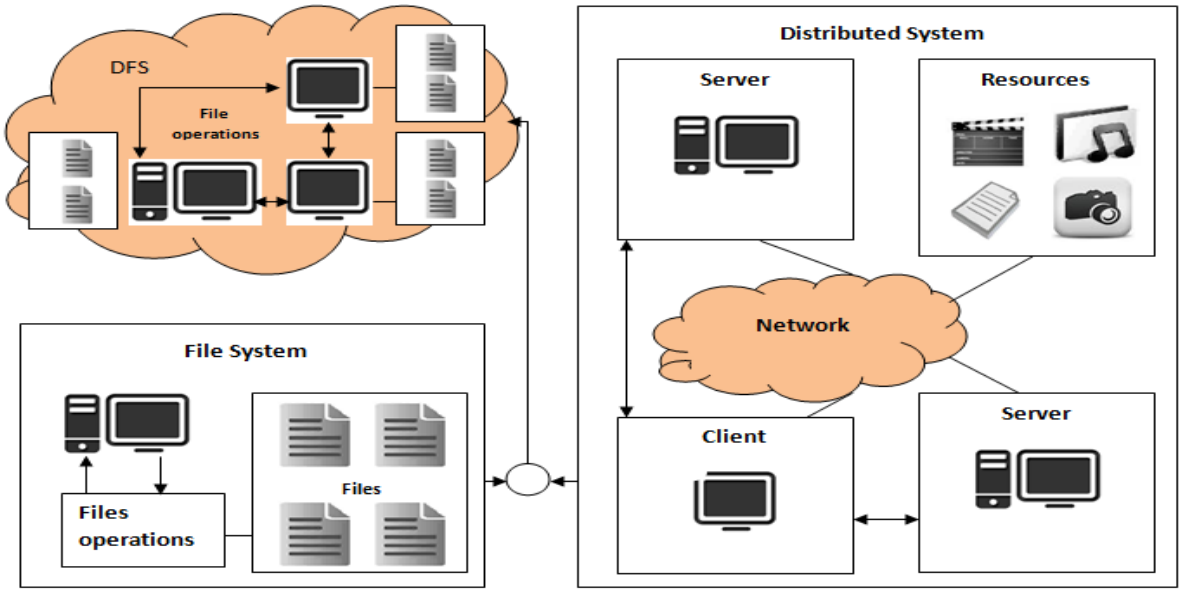


Figure 2: Diagram shows relationship between the Distribute System, File System and DFS

2.2 Anatomy of Distributed Application

According to [4] a distributed application is built upon several layers. At the lowest, a network connects a group of host computers together so that they can talk to each other. Network protocol like TCP/IP let the computers send data to each other over the network. Higher- level services can be defined on top of the network protocol, such as directory services and security protocols. At the top of these layers distributed application runs, using the mid – level services and network protocols as well as the computer operating system to perform tasks across the network.

According to the description above the distributed application can be broken down into following parts:

- **Process** – a process is created by describing a sequence of steps in a programming language while the code is compiled and run on computer. A process is used to perform tasks such as file sending and receiving over the network. The process usually has access to the resources of the computer such as CPU or I/O devices though the operating system.
- **Threads** – every process has at least one thread of control and every thread can run independently from other threads. The purpose of thread in distributed application is to monitor for example input from a socket connection such as mouse movements etc.),and provides feedback to the user through outputs devices (monitor, speakers, etc.).
- **Objects** – programs written in object – oriented language are made up of cooperating objects. Also the process in applications can be made up of one r more objects, and these objects can be accessed by one or more threads within the process.
- **Agents** – usually refers to the functional elements of a distributed application and agent is high – level system component. For example a banking application can be broken into a customer agent, a transaction agent or information service agent.

2.3 Processing Application

The idea behind the DFS is that the resources of a specific machine are treated locally to itself, whereas the resources of the other machines are remote. A DFS manages a collection of dispread storage device (**Figure 3**). The overall storage space of DFS consists of locally and remotely located storage spaces. Those storage spaces are managed by DFS correspond to several sets of files. A

component unit is the smallest set of files that can be stored on a single machine, independently from other units [18]. All files from the same component unit must reside in the same location.

Most of current DFS is constructed on client/server architecture, and it permits client to access and process files stored on the server as if they were stored locally [18]. Here it mentioned three key terms:

- **Service** – in DFS a service is a particular type of function that some server provides to clients
- **Client** – client is a process that makes request of service from other programs
- **Server** – server is the service management software running on a specific machine used to fulfil and wait for service client in the local computer or remote computers

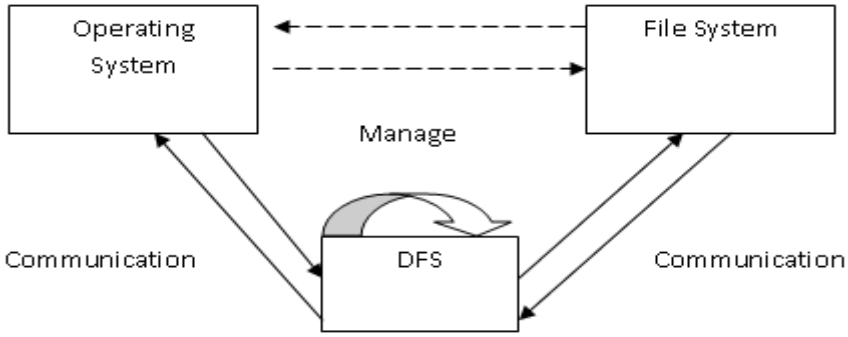


Figure 3: Communication managed by DFS

2.4 Naming and Transparency

Naming is a mapping between logical and physical objects. In DFS, users represent a logical data object with a file name that is a textural name in user-level, but the system physically stores data blocks on disk tracks which is numerical identifiers mapped to disk blocks in system – level. These two levels mapping provides users with an abstraction of a file that hides the details of how and where the file is stored on a disk. In DFS, the location of the file in the network is added to the abstraction. In conventional file system, the range of the naming mapping is represented as an address in a disk [18]. The file abstraction leads us to the notion of file replication (**described in section 2.5 File Replication**). When we want to access a specific file name the mapping returns a set of locations of the file replicas (**Figure 4**).

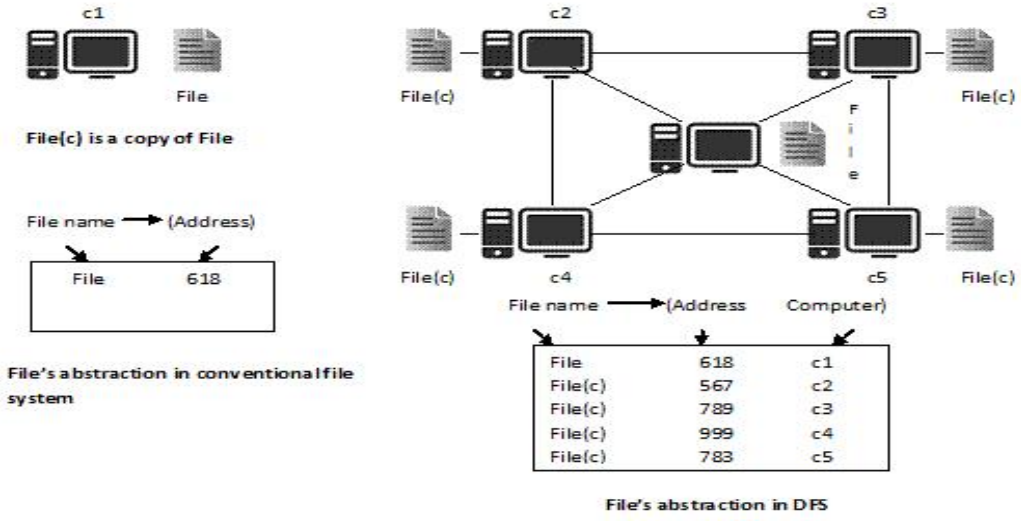


Figure 4: Abstractions in conventional file system and DFS

But, this abstraction allows hiding both their locations and existence of multiple copies (Figure 2). There are two notion related mapping in DFS:

- **Location transparency** – the name of a file does not reveal any hint as to its physical storage location [4].
- **Location independence** – the name of a file not be changed when the file's physical storage location change [4].

Both definitions are relative to naming of files. According to [4] location independence is a stronger property than location transparency because it can map the same file name to different locations at two different instance of time.

2.5 File Replication

File replication is a vital concept in all distributed systems. In a system that supports replication, a file may be represented by several copies of its contents at different location. Replicas of file are useful redundancy for accessing wait and delay of files because it helps to share the loads between servers. If a server copy is not available at some point, clients may simply request that file from a different server.

Almost all distributed systems uses file replication and backup. However, we have to notice that file replication is more powerful feature because there is a key difference between those two. Since backup create checkpoint in the file system to which the system can be restored that file replication is a policy of keeping redundant copies of a file. This replication policy makes sure that files are accessible even if one or more components of DFS fail.

There are number of techniques for file replication that are used to maintain data consistency. The replication service hast to maintain all the replicas having the same versions of updates. This is known as maintaining consistency or synchronization [3]. Replication techniques can be divided into three main classes: explicit replication, lazy file replication and group file replication [18].

- **Explicit replication**: The client explicitly writes files to multiple servers. This approach requires explicit support from the client and does not provide transparency [18].
- **Lazy file replication**: The server automatically copies files to other servers after the files are written. Remote files are only brought up to date when the files are sent to the server. How often this happens is up to the implementation and affects the consistency of the file state [18].
- **Group file replication**: write requests are simultaneously sent to a group of servers. This keeps all the replicas up to date, and allows clients to read consistent file state from any replica [4].

2.6 Synchronous and Asynchronous Message Passing

Message passing is a form of communication used in object oriented programming, and inter- process communication. In this model, processes or objects can send and receive messages or other processes. By waiting form messages, processes can also synchronize [9].

In synchronous message passing (**Figure 5**) sender and receiver has to wait for each other to transfer the message. That is, the sender will not continue until the receiver has received the message [15]. The synchronous communication has two advantages:

- Program can be simplified (there is a synchronization point between sender and receiver on message transfer)
- No buffering is required (message can be always stored on the receiving side)

In asynchronous message (**Figure 6**) passing the message is delivering from sender to receiver without waiting for the receiver to be ready [6]. The advantage of asynchronous communication is a follow:

- Sender and receiver can overlap computation because they do not wait for each other

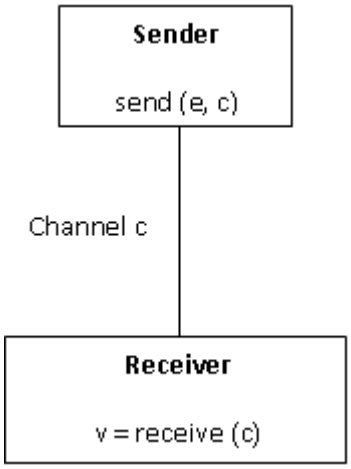


Figure 5: Synchronous message-passing channel

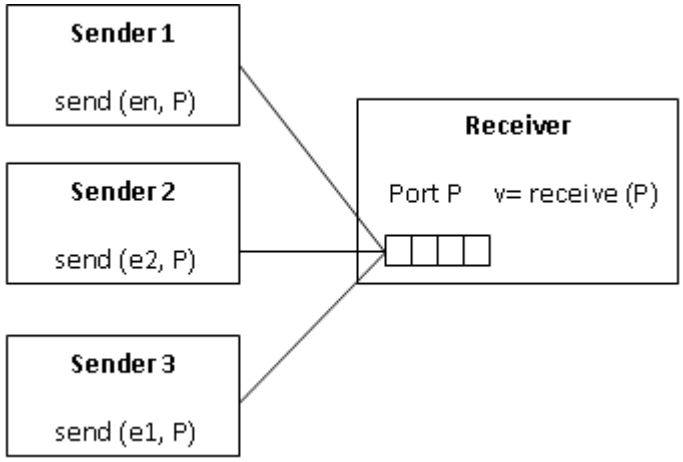


Figure 6: Asynchronous message-passing channel

2.7 Protocol

Protocol is a set of rules to control the data (files) communication in a DFS in order to perform the given task [8]. A protocol can be treated as a kind of agreement or procedure rules about the exchange of file message in a DFS. Additionally to this the protocol determines whether the network uses a peer-to-peer or client-server architecture. If an agreed-upon format data is transmitted between two devices then the protocol determines the following:

- How a sender indicates to finish sending a file
- How a sender indicate to finish receiving a file
- Error type checking mechanism
- Data compression method

Now we have some sense of how this protocol is turned into a distributed file system and its importance in client/server communication. Let us also consider a simple application which reads a file. In this application the client side tracks open files, and translates application request into set of protocol messages. The server simply responds to each protocol message in order to complete the client request. In DFS system protocol plays the same role, and provides sufficient communication between client and server.

In general the Internet and computer networks use protocols very extensively. Some of the protocols are very simple but others are complex and intellectually deep. The DFS is a very complex system itself so protocol will play crucial role in effective communication.

The purpose of this chapter is to answer: why the methodology is so important in software building process? Well according to [4] the software engineering is the practice of using selected process techniques to improve the quality of software development effort. This is based on the assumption, that a methodical approach to software development result in fewer defects and, therefore, ultimately provides shorter delivery times and better value.

The challenge in selecting and following a methodology is to do it wisely – to provide sufficient process disciplines to deliver the quality required for project success, while avoiding steps that waste time [4]. One system development methodology is not necessary suitable for use by all projects. I chose two methodology techniques which are best suited to this project, and I described them below.

3.1 Boehm Spiral Model

Boehm Spiral model was chosen for the development of the system. The spiral model uses incremental development, with the aim managing risk. In the spiral model, developers define and implement features in order of increasing priority. An initial version of the system is developed, and then repetitively modified based on input received from evaluations. The development of each version of the system is carefully designed using the steps involved in. With each iteration around the spiral (beginning at the centre and working outward), progressively more complete version of the system are built [4].

A key advantage of this approach is that the major difficulties are resolved at an early stage. In the event that some of the difficulties cannot be resolved using one approach may be pursued. In the event that the difficulties cannot be resolved at all, the project requirements may need to be modified so that a time effective and feasible solution can be developed.

This model was selected because as I mentioned above spiral model has some advantages which are best suitable for this project. Also, the model is versatile for the project dynamics and progressive development design. It is wise to build the various parts in a predefined order. This will give the project the versatility of a plug-in type project, meaning the ability to have core system and various add-on parts. This way it is much easier to test the various parts separately.

The project requirements remain that the end product should be able to provide system with client-server functionality in the form of packages of classes. Following the requirements gathering phase, an initial static analysis took place. The products of this analysis were the initial class diagrams of the core of the Client package with the system for clients. During this phase, some of the class diagrams turned out to be erroneous, so they were plotted again. This procedure was repeated many times for the Client package until it reached an acceptable state. The same technique was used during the implementation of the remaining packages. This is how the idea behind the spiral model was used in the project.

3.2 Rapid Application Development-RAD

The idea of RAD was chosen to implement the client-server model for DFS. The challenge facing software engineers and development organization can be summarized as more, better and faster [Shari]. The RAD development path attacks these challenges on providing systems faster, while reducing cost and increasing quality.

The RAD methodology was chosen for this project because it works when the work can be broken into manageable chunks like in this project. The advantage is also that the model work fine when the project is developed in small team (1 person).

3.3 Prototyping

Prototyping was chosen to build the interface for client and server. The rationale for choosing this approach is discussed below.

A prototyping methodology is a software development process which allows developers to create portion of the solution to demonstrate functionality and make needed refinements before developing the final solution [4]. It is also excellent way to confirm understanding of the requirements and ensure that the proposed solution is consistent with business expectations.

The RAD prototyping is very effective way of eliciting requirements difficult to obtain from customers or by other means [4].

I chose this approach because prototyping methodology is suitable for implementing distributed systems, and can very quickly help confirm system navigation and other user interaction requirements. Prototyping allows the designer to obtain feedback from the users early in the project. During the project I develop several prototypes for the system after when I choose the best which matched the software specification.

3.4 System Development Technology

The DFS will be written in Java because Java is a powerful platform for developing distributed systems. That's why the project title is Java Distributed File System (JDFS). Java is one of the most popular programming languages used nowadays. Java provides all the necessary features for the advance network programming. Further study about Java suitability for distributed file systems is provided in Chapter 5: Java DFS.

The client and server will be coded in Eclipse because this software provides all advanced features for programmers and is delivered as Open Source software.

The peer-to-peer and client-server are two architectures used for distributed file systems. This chapter describes and perform analysis of two systems most widely used today: Sun's NFS and Hadoop DFS. The analysis mostly is based on the type of architecture used.

4.1 Sun's NFS

The idea of distributed file system was partially adopted by Sun Microsystems in developing their Network File System (NFS) for Unix-like operating system. This is a distributed file system for mainly local area networks-LAN) [11]. Clients in such distributed system have transparent access to remote file on a LAN under Unix. Today NFS implementation can be applied in almost all operating systems and computers including non-Unix systems, such as Microsoft Windows, Macintosh or MS-DOS.

In defining NFS, Sun took an unusual approach: instead of building a proprietary and closed system, Sun instead developed an open protocol which simply specified the exact message formats that clients and servers would use to communicate [12]. Then different groups could develop their own NFS servers and thus compete in an NFS marketplace. This approach works fine and today there are many companies that are selling NFS servers such as: IBM, EMC or NetApp.

4.1.1 Sun's NFS Architecture

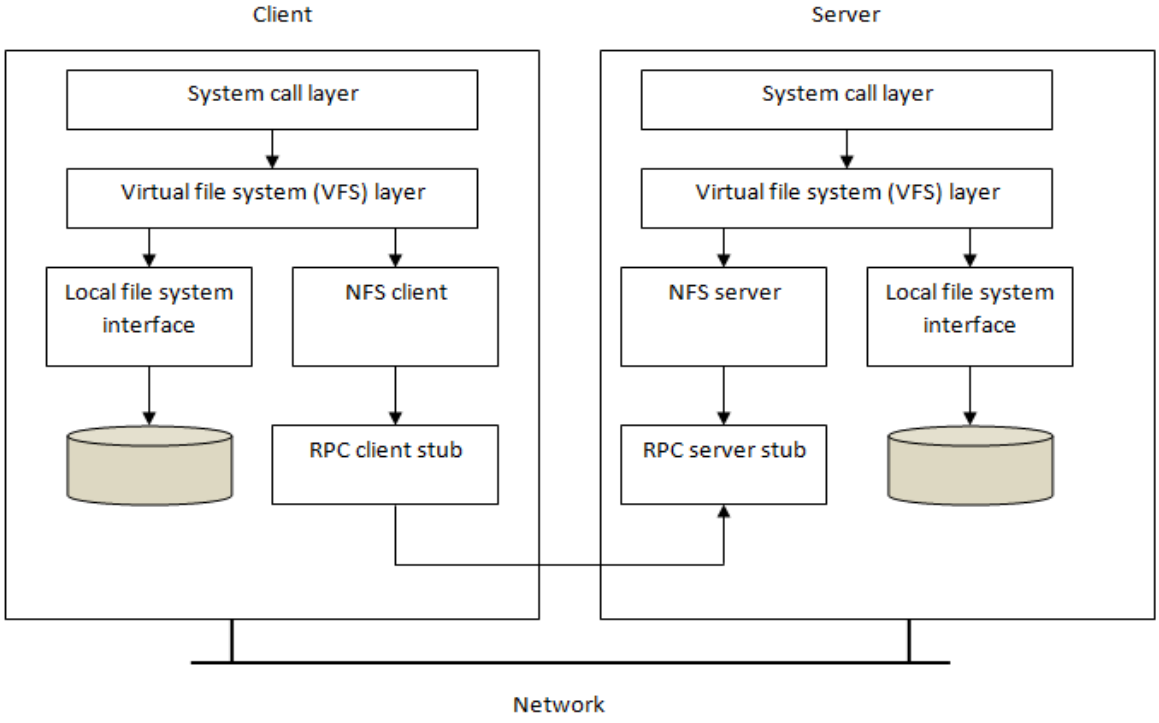


Figure 7: The Sun NFS architecture

4.1.2 Protocol

This section discusses the classic NFS protocol. Versions of different protocols are not considered here because most of the changes are shared by all versions.

NFS protocol provides a set of RPC's (Remote Procedure Call) for remote file operations and they are as follow:

- Looking up a file within directory – views the files as a root what provides the transparent access to all directories and files.
- Manipulating links and directories
- Creating, renaming, and removing files – clients can create and remove files as specified in their access rules.
- Getting and setting file attributes - allows getting and setting different set of attributes to different files: rename, delete, write,
- Reading and writing files – allows files to be read and write by clients.

4.1.3 Advantages of Sun's NFS

The Sun NFS was initially designed to meet the following design goals:

- Access transparency – the NFS API is identical to that used by local operating system.
- Scalability – NFS can be built to handle very large real world loads in efficient manner.
- File replication – read only replicas are supported
- Hardware and OS heterogeneity – NFs is implemented on nearly all known operating systems and hardware platforms.

4.1.4 Disadvantages of Sun's NFS

Despite numerous advantages, Sun's NFS has also some disadvantages and they are as follow:

- Migration transparency is not fully achieved within NFS, so it does not fully support mobile devices.
- Files replication with updates is not supported within NFS.
- Location transparency is affected because NFS does not enforce a single network so each client sees the file system their way.

4.2 Hadoop DFS

Hadoop comes with a distributed file system called HDFS, which stands for Hadoop Distributed File System, and provides framework for the analysis and transformation of very large data sets. An important characteristic of Hadoop is the partitioning of data and computation across many (thousands) of hosts, and executing application computations in parallel close to their data [7].

4.2.1 Hadoop DFS Architecture

An HDFS consist of a single NameNodes, a master server that manages the file system namespace and regulates access to files by clients. In addition, an HDFS consist a number of DataNodes, which manage storage attached to the nodes that they run on. In HDFS a file system namespace allows user data to be stored in files. Internally, a file is split into one or more blocks and these blocks are stored in a set of DataNodes. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. The DataNodes are responsible for serving read and write request from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode. The **Figure 8** below shows the basic architecture of HDFS.

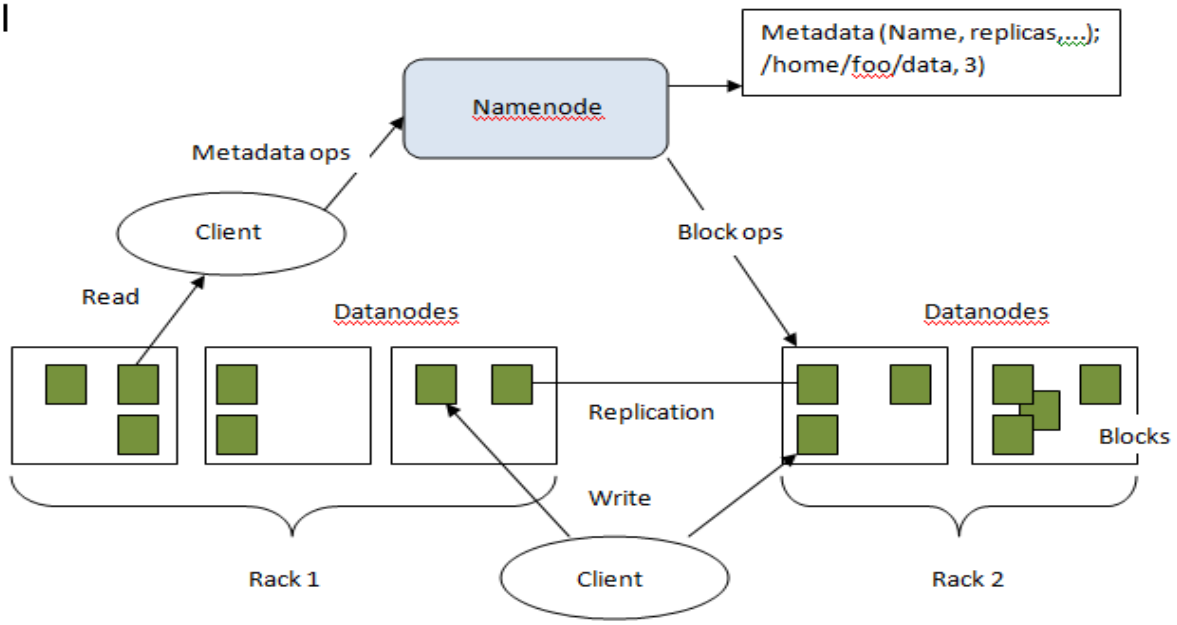


Figure 8: HDFS architecture

4.2.2 Advantages of Hadoop DFS

As mentioned above the HDFS is designed for storing very large files with streaming data success patterns, running on clusters of commodity hardware. The more detailed explanations for those properties are given below:

- Very large files – it means that we can operate with files that are measured in hundreds of megabytes, gigabytes or terabytes.
- Streaming data access – it means that HDFS is built around idea that most efficient data processing pattern is a write – once, read – many times pattern. A dataset is typically generated or copied from source, and then various analysis are performed on that dataset over time [14].
- Commodity hardware – it doesn't require expensive hardware to run properly. It's designed to run on cluster of commodity hardware for which the chance of node failure across the cluster is high, at least for large clusters [14].

4.2.3 Disadvantages of Hadoop DFS

Despite numerous advantages, Hadoop has also some disadvantages and they are as follow:

- Hadoop is less than ideal for some of the mobile aspects of mobile-cloud computing. It implements much of the functionality that our platform requires, but it does not cover all of the requirements.
- Hadoop is not conservative in CPU and memory usage because originally was designed for I/O bound jobs, i.e. those in which reading, writing, and transferring data are the most time-consuming operations.
- Hadoop uses technologies that are not well suited for mobile devices.

A feasibility analysis involves a detailed assessment of the need, value and practicality of a proposed enterprise, such as systems development [12]. In Software Engineering, feasibility analysis is used for definition of the system and its business requirements. It also identifies the essential risks associated with the project that must be addressed. Different project request decides different process and format for the feasibility analysis, In JDFS, it includes three techniques: technical feasibility, operational feasibility and Economic Feasibility. The relationship among them shows in **Figure 9**.

5.1 Technical Feasibility

Technical feasibility is the first technique in the feasibility analysis of JDFS. It essentially concentrates on the extent which JDFS can be successfully designed, developed and implemented

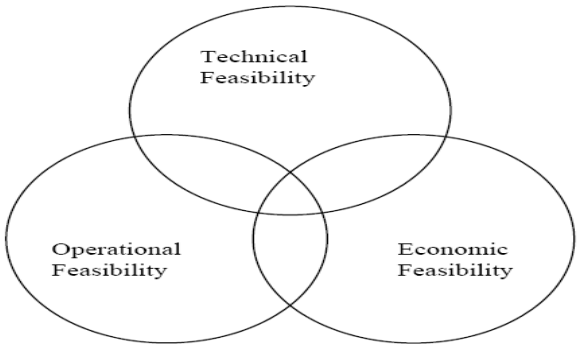


Figure 9: Relationship of feasibilities

For this purpose, it is necessary to analyze the technical features of JDFS expected to satisfy requirement in Chapter 4.

- Followed the trace of conventional client server DFS, JDFS realized the communication among clients with servers in network
- JDFS will be developed on Java platform with Java's Object-oriented and Network-oriented application techniques.
- Based on conventional DFS, JDFS will be applied on different operating environment, such as Windows, Linux, and Mac OS, etc.
- JDFS also realizes portability which is one of most significant features of software engineering design. From installation to implementation, it ensures that users feel easily and comfortably during file operations.
- Extensibility and saleability also should be considered in JDFS. For this goal, the Java techniques will be illustrated in the design of JDFS.
- Because of the apparent of simultaneously data passing, Java concurrency communication techniques application will be found in design process.

5.2 Operational Feasibility

Operational feasibility meets the requests and expectations of users. To JDFS, user acceptance is an important determination of operational feasibility. It requires careful consideration of:

- **Corporate background knowledge** means that JDFS can runs on different operating system, so it is suitable for different users who are familiar with different operating environment.
- **Effective communication** means user can access files stored in other computers immediately as his/her request is sent out. Vice versa, if other computers send request to the user, user's computer responds the request as soon as possible.

- **Free toolkit** is that the system is economical to be developed. Its design environment is based on Java Runtime Environment that is free to download from Sun company website, so is J2SDK.
- **Performance** is that the design and implementation of JDFS should avoid the network bottlenecks, so that network traffic is maintained at proper level. Therefore, the users can maintain their JDFS effectively at a user level.
- **Information** is stored accurately in JDFS however the computer is on-line or not. The users can manipulate the files operation in a friendly user interface.
- **Security** is one of the most important factors in a system development. JDFS allocates username and passwords to each computer in the network. Accordingly, if users want to access the data shared in JDFS, they must log in certain specific computer by the authorized identity.

5.3 Economic Feasibility

A systems development project may be regarded as financially feasible or good value to the organisation if its anticipated benefits outweigh its estimated costs. The costs and benefits for economic feasibility can be break down into four categories: development costs, operational costs, tangible benefits and intangible benefits. The potential cost and benefits were identified and they are showed in **Table 1** below:

Development Cost
Development Team Salaries £500,000
Hardware and Software £150,000
User Installation £0
Data Conservation Cost £3,000

Operational Cost
Software Upgrades £2,000
Hardware Upgrades £10,000
Operational Team Salaries £40,000
User Training £2,000

Tangible Benefits
Increased Sales ≥ £300,00
Reduction in Staff ≥ £10,000
Reduction in IT Cost ≥ £30,000

Intangible Benefits
Increased Brand Recognition ≥ £100,000
Higher Quality Products ≥ £40,000
Improved Customer Services ≥ £80,000

Table 1: Economic feasibility analysis in order to see if the project is worth to invest or not

Development Costs + Operational Costs = £707,000

Tangible Benefits + Intangible Benefits = £290,000

Because some cost will be reduced or saved in the second year, such as Hardware and Software cost, User Training cost, which makes the benefits stay or exceed the current cost next year. Obviously, the benefits will overweigh the cost of JDFS development in two or three years. Therefore, the development of JDFS is valuable business for users and organizations.

The original Java motivations were concerned mainly with reliability, simplicity and architecture neutrality. The Sun Microsystems was developing Java seeing it as a programming language supporting networking, security and multithread operations. All these features make Java powerful development environment for distributed systems.

Java Distributed File System (JDFS) is one of the most powerful systems based on the Java platform. JDFS can be treated as a derived DFS from conventional DFS from which it inherits some features in nature such as:

- **Transparency –** the logical storage locations of shared files and directories are not revealed because the users only access the shared files and directories by their logical name.
- **Synchronous message passing –** when remote shared file or directory is modified by certain user, the machine can't accept any more other processes to the machine where the file is stored. This is achieved by channel blocking between those two machines.

In this section, I review some of the features of Java that are particular interested in distributed applications. Due to the design of JDFS is based on Java platform and application techniques it also inherits the characteristic from Java, and they are as follow:

- **Object – Oriented –** according to [15] object technology is based on the assumption that human cognition relies on the perception of reality in the form of objects – objects that have characteristic properties and show specific behaviour, that are related to each other, and that can interact amongst themselves. From an object – oriented software development perspective, an object is the IT-suitable representation of real – world object obtained by applying the principle of abstraction. Based on this definition each of machine, client, file and directory can appear as an independent object in JDFS. Each of those object have their own methods and variables which affects the data communication and system execution in JDFS.

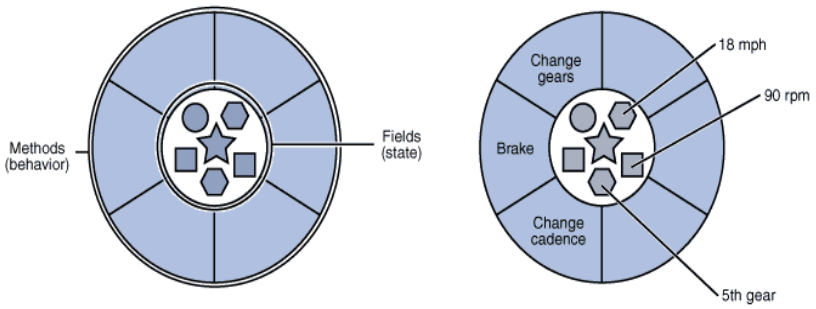


Figure 10: A software object and bicycle representation according to object – oriented concept (diagram downloaded from http://download.oracle.com/javase/tutorial/java/concepts/)

- **Abstract Interfaces –** is a valuable tool in developing distributed systems because interface describes the operations, messages and queries a class of objects is capable of servicing, without providing any information about how these abilities are implemented [15]. If a class is declared as implementing a specified interface, then the class has to implement the method specified in the interface. This approach has a couple of advantages:
  - Agents in the system can talk to the specified interface without knowledge about how this particular interface is implemented in a class.
  - The class implementation can be changed as needed.

- The class can be moved to the local host then the local implementation of the interface can act as a surrogate or stub. And the calls are forwarded to the interface over the network to the remote class.

- **Platform Independence –** Java stand for programming language and as well can be called platform so it allows code to be compiled into platform – independent byte – codes using Sun's Java compiler or any other available compilers on the market. Java is an independent software application and suitable to any operating system such as Windows, UNIX or Mac.
- **Fault Tolerance Through Exception Handling –** Java supports throwing and catching errors and exception, both systems – defined and applications-defined. Handling an exception is a powerful tool in distributed system because it allows wrapping any potential exception – causing code with **try/catch/finally** statement, where each catch clause handles a particular type of exception. It often occurs that called method generates an exception. Then, an exception is going to be handled by a **catch** clause in a calling method. Whether, the try block runs the code without problem, or exception gets thrown, but the code in the **finally** block is always called. The purpose of **finally** block could be to clean up any resources created in the try block, but this is one of the examples using **finally**.
  Also, the Java use exception handling to characterize, diagnose, and potentially recover from them. It is possible, because an exception is represented as an object in Java environment, and it can be carried with data and other methods that can be used for specific tasks.
- **Network –** Java has strong support towards application that requires access to network. Java application can access object across network via URL as easily as access local files. Without a doubt the Java is one of the best choices for developing DFS.
  The **Figure 8** below shows a simple network application involving client and server. The communication model is very simple; the client sends commands to the server then the server executes the commands and sends commands to the client. This is one of the examples how Java can be exploited for distributed applications. When we are going to write the actual code our implementation usually includes the following elements:
  - A set of command objects to represent our command protocol between the client and server
  - A subclass of **java.io.DataInputStream** that understands out protocol
  - A client that can send commands in the right format to the server, and a serve that can accept client connection, read commands and send response back to the client

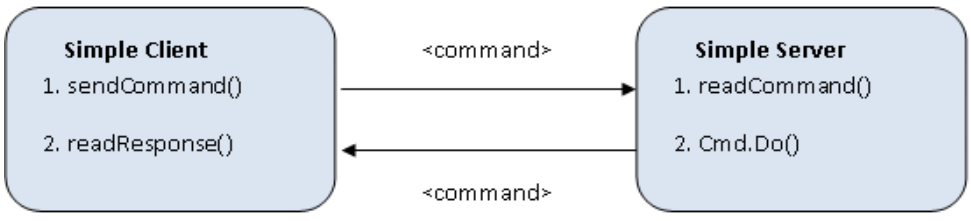


Figure 11: Simple Client/Server model

- **Security –** Java provides two kinds of security:
  - **Runtime environment –** means that Java Virtual Machine places restrictions on program operation and capabilities. It is allowed virtually no access to the local file system, very restricted network access, no access to local codes and libraries outside of the Java environment, and restricted thread manipulation capabilities.

- **Secure remote transaction –** means that Java provides capability to add user authentication and data encryption to establish secure network links, assuming that the basic encryption and authentication algorithm already exist. An example how this can be accomplish is to extend the **BufferedInputStream** and **BuffredOutpuStream** classes in java.io to authenticate and decrypt incoming data, and to sign and encrypt outgoing data.

Java itself provides secure network communications but also Java can be extended in order to provide more secure communication in applications where security plays crucial role.

- **Reusable –** Java provides number of packages of classes which can be used directly and frequently. Additionally, Java permits the users specify their own classes which can cooperate with Java's classes to meet user's request. Because the classes of Java can be reused and specified by users, so the time and effort of developers is reduced to minimum.
- **Multi – thread –** Multi – thread is one of the most important characteristic in Java systems, and it is always applied in real – time operating environment. Any class that we create can extend the **java.lang.Thread** class by providing its own implementation of run() method. When the thread is started, this run() method will be called and your class can do its work within a separate thread of control.
  By calling a Thread object's start() method a Java application tells the Java Virtual Machine (JVM) to start a separate thread of execution. Once started by the JVM, the thread exist until the run() method terminates. The thread concept is very important in every distributed system and becoming necessity in software development today. Threads are fast to set up, require less memory, and allow better encapsulation.
- **Feasibility –** Java programs can be easily modified and updated according to user's requirements.

A requirement is simply a statement of what the system must do or what characteristic it must have. From different point of view, a requirement can be broadly divided into user requirement and system requirement. User requirements, sometimes called business requirements, focus on business user needs. In design level, user requirements evolve to become more technical. It describes how the system will be implemented.

7.1 JDFS User's requirements

The JDFS system has to implement the following user's requirement:

- **Security:** Each user must log in his/her system by correct username and password, which guarantees security of local and remote shared files and directories. Also, the file with user's login details never leaves the local machine what make the security higher.
- **Communication:** JDFS is suitable for all of users who are familiar with different operating environment, such as Windows, Linux. So JDFS can be set up in the most of operating systems.
- **Accuracy:** In JDFS, the files and directories are updated in real time, wherever they are stored. So JDFS ensure users can access to the current shared data accurately during communication.
- **Constancy:** JDFS solve the trouble of data confliction in synchronous access, i.e. when a specific file or directory is being modified by user; others cannot make any modification on it, such as rename, edit, delete and cut.
- **Error Indication:** If a user makes some mistakes on file operation in JDFS, there will be a notice dialog with system message provided for leading user to correct his/her operation.
- **Friendly GUI:** Graphic User Interface of JDFS's design concentrates on two issues: readable and friendly. User can easily deals with file operations through the GUI components. User also can enjoy the remote and local shared data communication by friendly platform with other computers in JDFS.
- **Flexible Access:** Accessing to the remote shared files and directories is as easily and flexible as those happened in local machine. So users can be hardly aware of the file or directory that they access comes from remote machine in JDFS.

7.2 JDFS System Requirements

From the developer's perspective, requirements are usually called system requirement. Actually, there is no obvious distinction dividing a user requirement and a system requirement. Requirements evolve from detailed statements of the business capabilities that a system should have to detailed statement of the implementation in the technical way. System requirement can be break down into functional requirement and non-functional requirement in nature.

7.2.1 Functional Requirement

A functional requirement relates directly to information it needs to contain or a process the system has to perform. In JDFS, the most of functional requirements focus on the file/directory operations and system features.

- **Share:** A file/directory operation can be executed on local machine in JDFS. It is used for selecting directory to be shared folder of the shared recourse on local machine.
- **Cut:** A file/directory operation can be executed on local machine and remote machine, in which a user moves local or remote files/directories into buffer, awaiting the "paste" file

operation to accept them. Precondition is that the remote file needs 'cut' and 'delete' permissions, and nobody is using the file/folder now.

- **Copy:** A file/directory operation runs on local machine and remote machine, in which a user can copy a remote shared file or directory by the permission of "copy", and sends the copy of the file/directory into buffer, awaiting the "paste" file operation to accept them.
- **Paste:** A file/directory operation runs on local machine, where it moves the file or directory from buffer to the destinations.
- **Delete:** A file/directory operation can be applied on local machine and remote machine. If user wants to delete a local file or directory, "delete" operation can be applied directly. Whereas, the user must send requests to the computer in which the remote shared file/directory is stored. The host of the file/directory received the request and then delete the file/directory. The pre-condition of "delete" operation is that permission of the remote shared file/directory is "delete", and there is nobody access the file simultaneously.
- **Read:** A file operation is allowed on local machine and remote machine. Users can read the local files directly, while they must have "read" permission to read the remote shared files in JDFS. This operation allowed the multiple accesses to a remote shared file.
- **Write:** A file operation is applied on local machine and remote machine. User is able to edit the local files directly, whereas they have to get the permission of "write" to edit the remote shared files. The precondition is that nobody edits the same remote shared file synchronously or the file is being used.
- **Rename:** A file/directory operation runs on local machine and remote machine. User can rename local file/directory directly. Contrary to the remote shared file/directory, they have to send the request to the remote computer where the shared file/directory is stored. The host of the remote computer receives the request, and then rename the file.
- **Set Permission:** A file/directory operation is applied on local machine, in which a user is allowed to change the permission of the local share file/directory.
- **Path of resource:** A system feature is located on local machine. It is responsible for displaying the logical path of local shared resource and remote shared resource.
- **File Permission:** A system feature is addressed in local machine. It is capable of showing the permission of the remote shared file or directory at present.

7.2.2 Non-Functional Requirement

Non-functional requirements reflect the behavioural characteristics that the system must have, so it can be divided into operational requirements, performance requirements, and security requirements.

7.2.2.1 Operational Requirements

- JDFS will operate in Window, Linux, etc.
- JDFS is capable of reading and editing most of types of the file, such as Word documents, HTML, Text, Excel tables and so on.

7.2.2.2 Performance Requirements

- In JDFS, when local machine received requests from other remote machines, the responses will be sent back as immediately in order to void the bottleneck problem.
- The shared files and directories must be updated at real time, which ensures users can get the most current files or directories.

7.2.2.3 Security Requirements

- The correct username and password is the pre-condition of manipulation of JDFS. JDFS allocates unique username to each machine in workstation for the security of shared resource in network. The username and password never leave local machine.
- The shared files/directories are protected by the permission given by remote users in JDFS.

Functional models describe the interaction between user and the Java Distributed File System (JDFS). In object – oriented system development two types of models are used to describe the functionality of JDFS: use case diagram and activity diagram.

8.1 Use Case Diagram

According to [5] a use case diagram is used to model the interaction between a system and its internal entities (actors) in terms of use cases. The diagram below was designed as first UML diagram for the system because it helps better understand its intended behavior.

In JDFS I distinguished two actors whose play the crucial roles in distributed file system. The actors are as follow:

- **Local User** – user who can perform local file operation without limitation of file access permissions on the local machine in JDFS.
- **Remote User** – user who operates the files located on remote machine through file access permissions on local machine in JDFS.

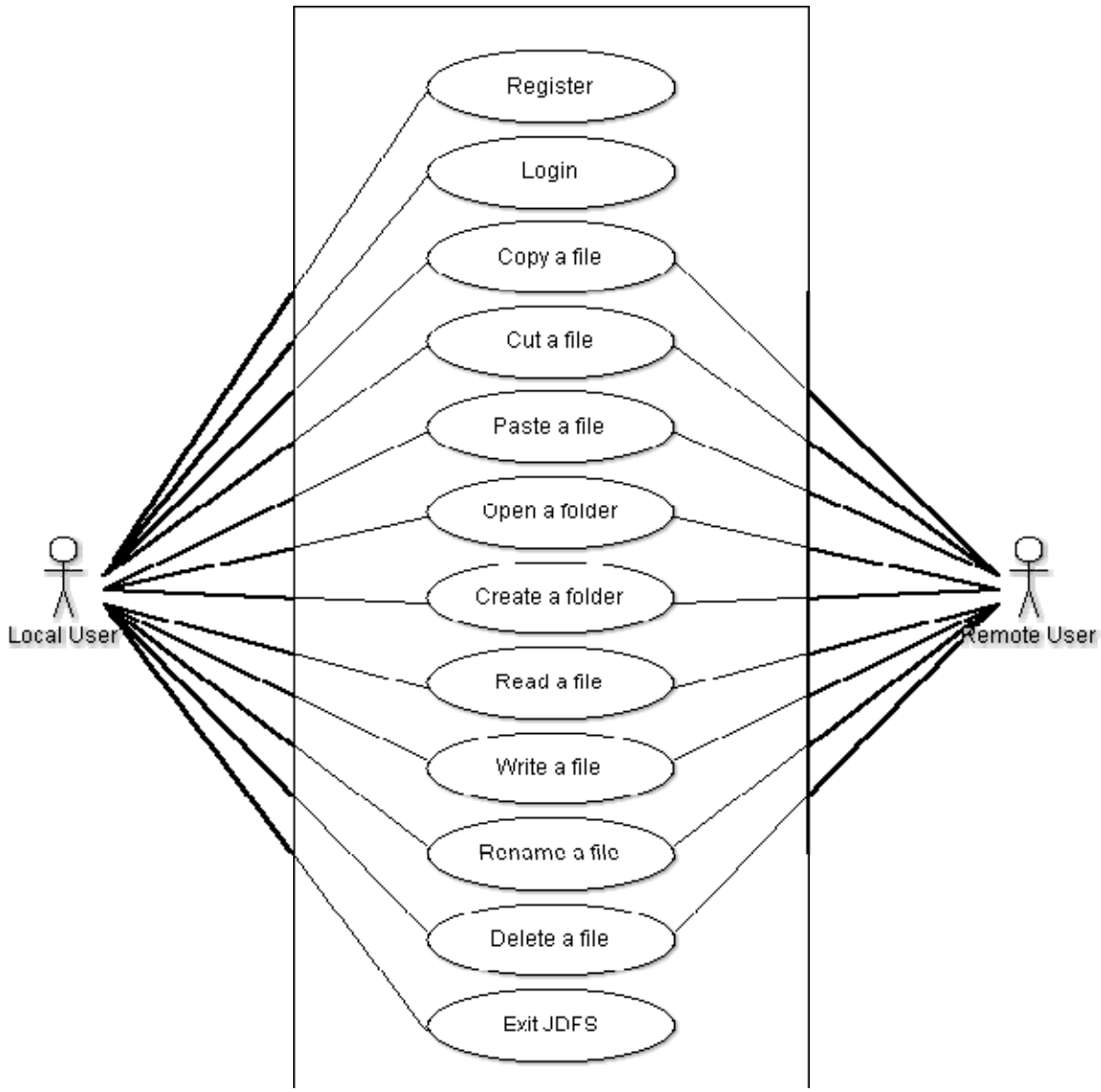


Figure 12: Use Case Diagram for JDFS

8.1.1 Detailed Use Case Diagram

Detailed Use Case Diagram was produced as well the best model the software specification in terms of user and systems interaction.

a) Use Case Name: Register

Actors:	Local User
Description:	User need to register to the server in order to obtain username and password
Precondition:	None
Basic flow of events:	1. The system displays a registration form to user. 2. The user fill the form in and then press a submit button. 3. The form is validated in term of required data. 4. The system display confirmation dialog and link to login site.
Post condition:	The system provide link to login window.

Table 2: Table shows detailed use case description for 'registration' use case

b) Use case Name: Login

Actors:	Local User
Description:	User need to login to the system in order to be able to upload and download file.
Precondition:	None: User is registered
Basic flow of events:	1. The system displays a registration form to user. 2. The user fill the form in and then press a submit button. 3. The form is validated in term of required data. 4. The system displays confirmation dialog and link to login site.
Post condition:	The system provide link to login window

Table 3: Table shows detailed use case description for 'login' use case

c) Use case: Read and Write a file

Actors:	Local User, Remote User
Description:	User need to login to the system in order to be able to see all files.

Precondition:	None: User is logged in to the system
Basic flow of events:	1. The system gives the same view of server's file to all users. 2. The can try to read or write the file. 3. The system display an information dialog if the user has right to read or write the file
Post condition:	None

Table 4: Table shows detailed use case description for 'read and write' use case.

d) Use case: Copy/Paste/Rename/Delete a file

Actors:	Local User, Remote User
Description:	User need to login to the system in order to be able to see all files.
Precondition:	None: User is logged in.
Basic flow of events:	1. The system gives the same view of server's file to all users. 2. The system displays buttons in order to copy, paste, delete or rename a file. 3. The system display an information dialog if the user has no right to copy, paste, rename or delete a file.
Post condition:	None

Table 5: Table shows detailed use case description for use 'copy/paste/rename/delete a file' use case.

e) Use case: Open a folder

Actors:	Local User/Remote User
Description:	User can open any folder from disk or server
Precondition:	None: User is logged in.
Basic flow of events:	1. The user double clicks on a file. 2. The system opens the file and the user can view the file.
Post condition:	None

Table 6: Table show detailed use case description for 'open a folder' use case.

f) Use case: Crete a folder

Actors:	Local User/Remote User

Description:	User can create a folder on local disk or on server
Precondition:	None: The user is logged in.
Basic flow of events:	1. The user press the button 'create directory'. 2. The system confirms the operation by displaying dialog with information that folder was created or not.
Post condition:	None

Table 7: Table show detailed use case description for 'create a folder' use case.

g) Use case: Exit JDFS

Actors:	Local User
Description:	The user is able to exit the system any time
Precondition:	None: The user can see the system's interface.
Basic flow of events:	1. The system gives the exit button in the right top corner of system's interface. 2. The user press the exit button and system display dialog if the user is sure to exit the system.
Post condition:	None

Table 8: Table show detailed use case description for 'exit' use case.

Chapter 9: Structural and Behavioural Modelling

9.1 Class diagram

Class diagram model the classes or building blocks used in a system [5]. Class diagram illustrates the specification for the software classes and interfaces in an application. Typical information includes in a class diagram are (a) classes, associations and attributes, (b) interfaces, with their operation and constants , (c) methods, (d) attributes type information, (e) navigability and (f) dependencies.

The class diagram also shows the association and navigability that are necessary to the class diagram. And in the class diagram each attribute is fully typed appropriate data type.

9.1.1 Overall System Decomposition

The diagram in **Figure 9** describes the overall system decomposition and layering. Many of the subsystem in JDFS have been identified properly during analysis stage. Those that have been clearly identified are the **dist_client**, **dist_server**, **dist_file**, **dist_system** and **dist_communication**.

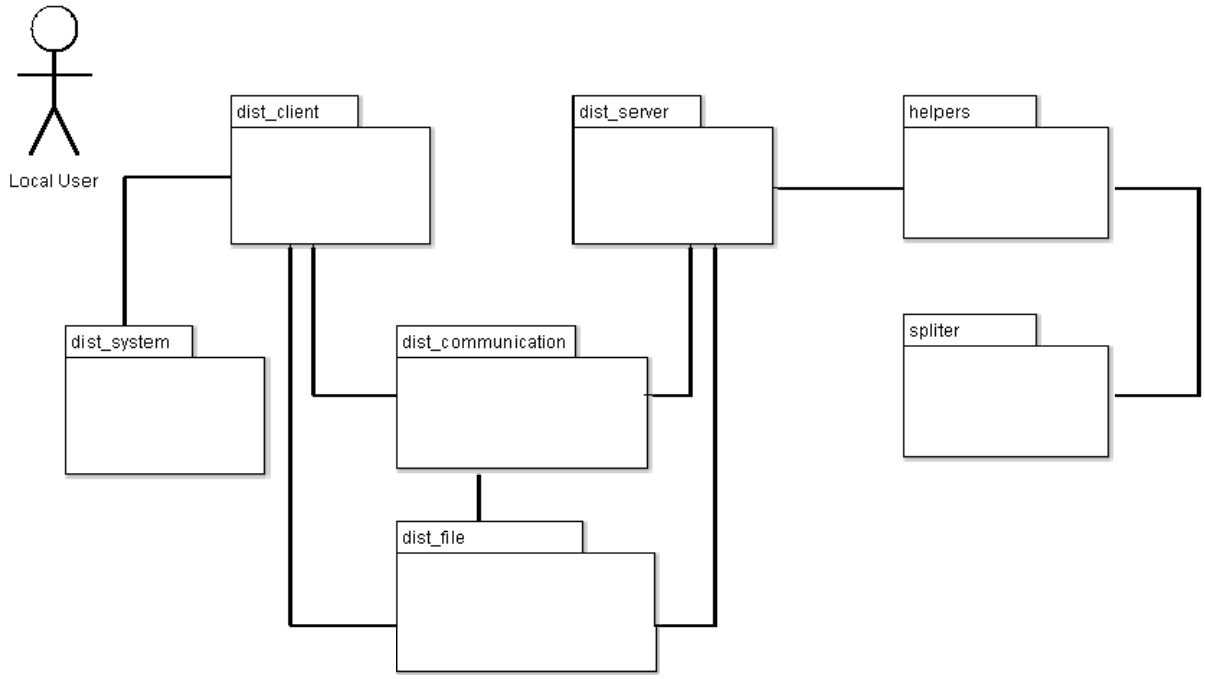


Figure 13: Overall system decomposition showing the main packages and relationships among them

The main packages identified with their functionality description are provided below:

- **dist_client** – package keeps client's classes. They are mainly responsible for displaying GUI to system's users such as registration, login and main system window.
- **dist_server** – package keeps the server's classes. It is mainly responsible to run the server program in machine background.
- **dist_system** – package keeps classes which are responsible for controlling access to the system by different users.
- **dist_file** – package keeps classes which perform operation on files such as: control the access to files by different users, perform file replications or specify the file rights.
- **dist_communication** – package keeps all classes responsible for sending and receiving massages between client and server.

However, to the main packages identified before I added two more, because future consideration required it. The packages and their functions are as follow:

- **helpers** – package keeps one class which is responsible for producing and gathering servers logs. Logs will be written in a doc file called serverLogs. The file can be viewed in order to know what is going wrong with the server operations.
- **splitter** – package keeps two classes responsible for file splitting from different servers. As we know files has to be spread into several pieces across different servers in order to protect the files against server failures.

a) Client Class Diagram – dist_client

The client portion of a system corresponds to the system run by individual users. Because system user interacts with the client, the client program is usually much fancier than the server in regards how information is displayed. The basic responsibility of the client is connect to the server and

communicate the user's actions, along with receiving state information from the server and updated itself accordingly. The Class diagram for Client is showed below:

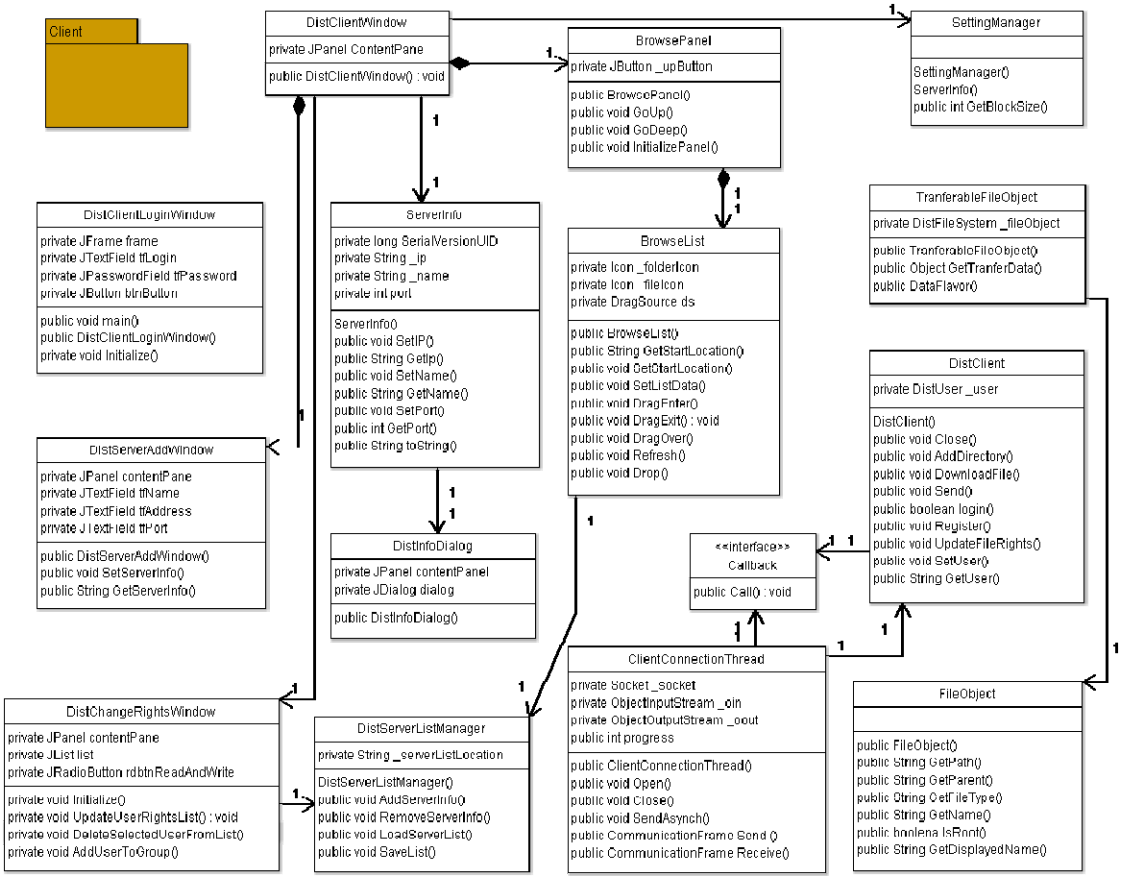


Figure 14: Client Class Diagram

b) Server Class Diagram – dist_server

In any system using a network, the server side acts almost like a referee, managing the different users and helping them communicate effectively. The JDFS server takes on the role of handling the network connection for each user, along with querying for and responding to events for users. The role of a generic server can be broken down into the following action.

1. Initialize the server socket
2. Wait for the client to connect
3. Accept the client connection
4. Crete a daemon thread to support the client
5. Go back to step 2

The class diagram for Server is showed below:

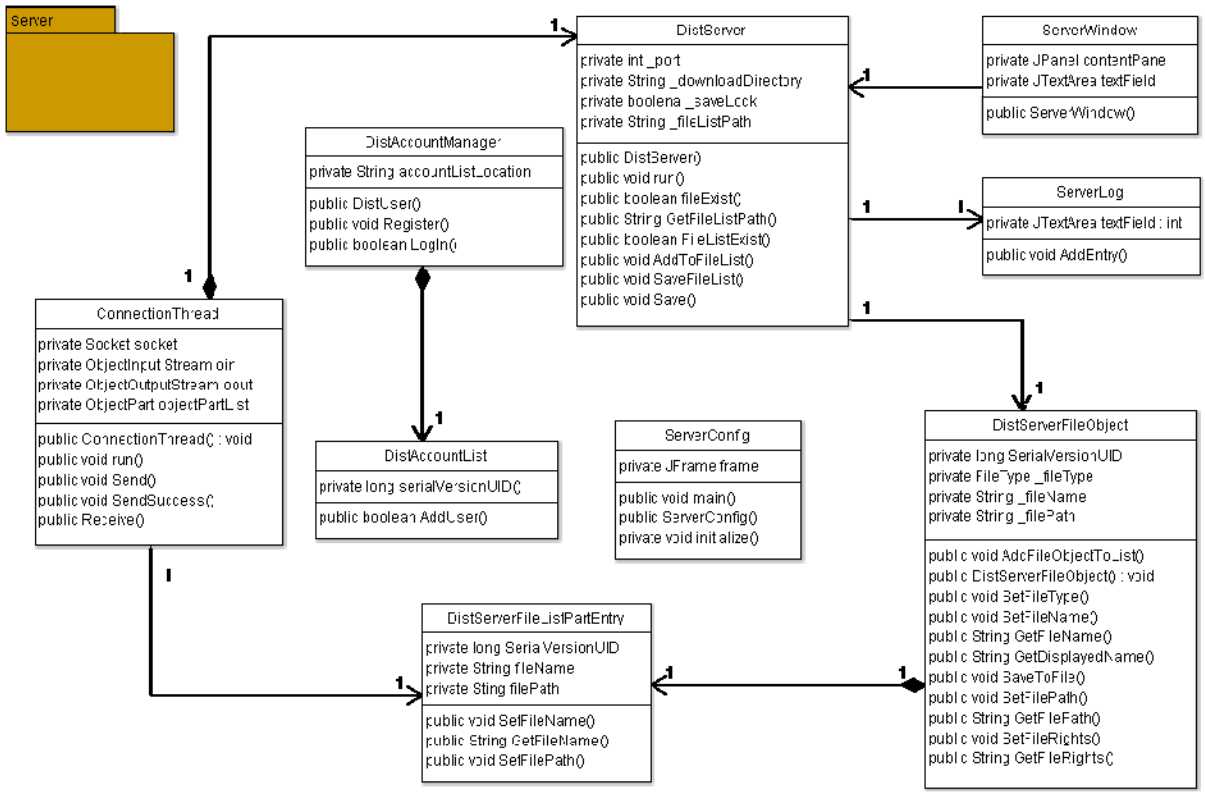


Figure 15: Server Class Diagram

c) System Class Diagram – dist_system

The dist_system portion of the system is responsible for the overall system security. It controls and checks the user's login details and user's rights to files.

Figure 16: System Class Diagram

d) File Class Diagram – dist_file

The dist_file portion of the file corresponds to the mechanism which allows users to see the same context of the files. It allows user to read and write files and also performs the copy, paste, rename and delete operations.

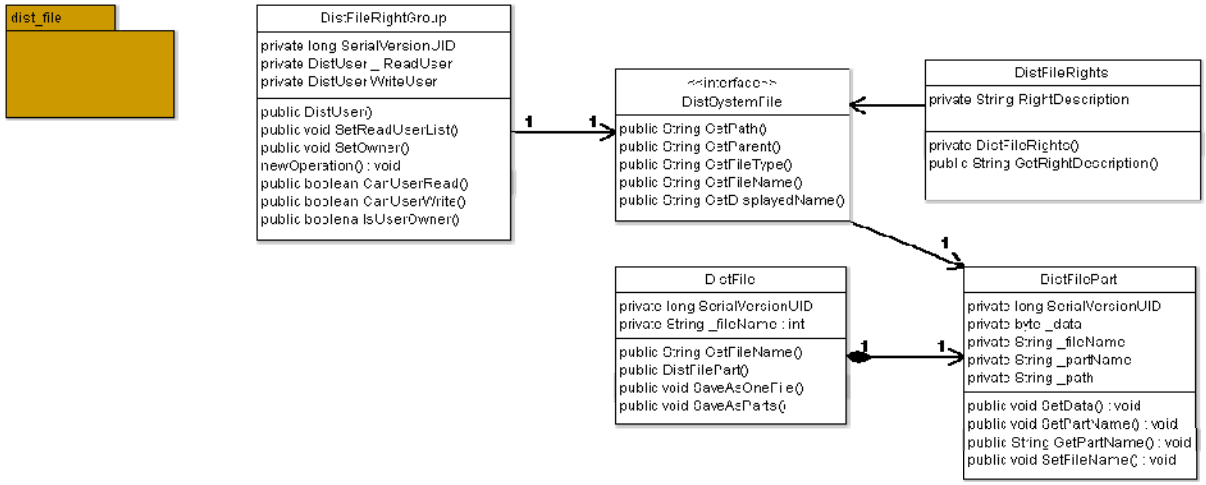


Figure 17: File Class Diagram

e) Communication Class Diagram – dist_communication

The dist_communication part of the system corresponds to the actual communication between the client and server. It allows to swap messages between client and server.

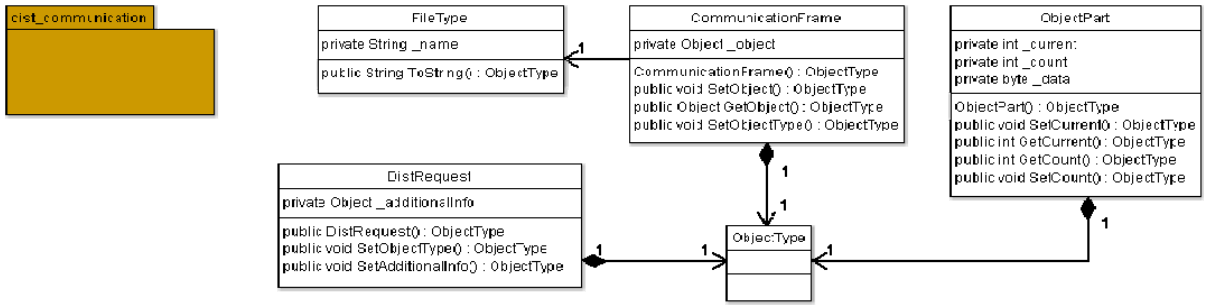


Figure 18: Communication Class Diagram

f) Helpers Class Diagram – dist_helpers

The dist_helpers part of the system is responsible for generating a log file with all error produced during fatal system's operations.

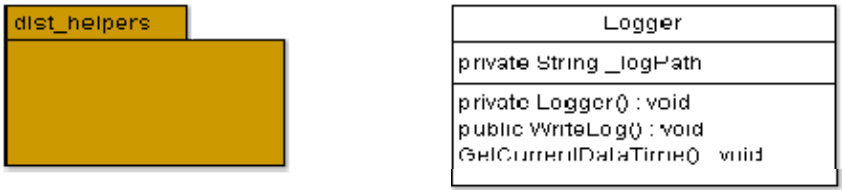


Figure 19: Helpers Class Diagram

g) Splitter Class Diagram – splitter

The splitter part of the system is responsible for splitting pieces of the same file into one file, because all files were divided into several pieces and distributed to several servers.

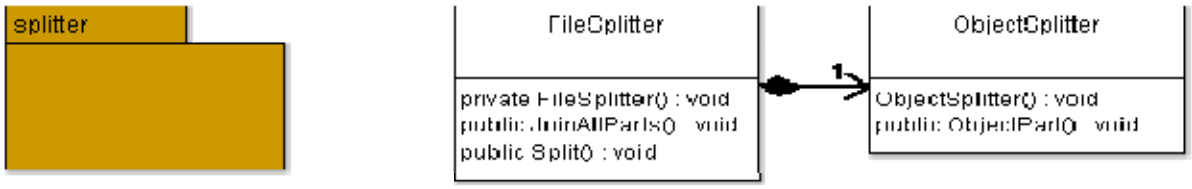


Figure 20: Splitter Class Diagram

9.2 Sequence diagrams

Behaviour modelling is to depict the behaviour of objects of the system such as interactions, events, control and data flow. Sequence Diagrams illustrates the objects that participate in a use case and the message that pass between them over time for on use case. The Sequence Diagram consist the following elements:

a) **Object** – participate in a sequence by sending and receiving the message.
b) **Lifeline** – represents the life of an object during a sequence.
c) **Message** – sends information from one object to another one.
d) **Execution occurrence** – represents when the object sends or receives message.

For the purpose of this report sequence diagrams were drawn and attached in the appendices (**See Appendices C-J**). The following sequence diagrams were produced:

- Connection Sequence Diagram
- Disconnection Sequence Diagram
- Delete/Paste/Rename Sequence Diagram
- Copy/Paste file Sequence Diagram
- Delete/Rename Sequence Diagram
- Read a file Sequence Diagram
- Write a file Sequence Diagram

Chapter 10: Interface Design

Design is more creative step than analysis because it is time to create the architecture of the program where functions and operations are described in detail, including screen layouts, process diagrams and

other documentation. In the design I will describe all the software futures and diagrams in details in order to develop the software with minimal additional input.

10.1 User Interface Design Methodology

The design methodology was chosen having in mind the system specification. These following methodology described in this section were used in JDFS development.

10.1.1 Prototyping and Design Process

The user interface design involved an iterative process of sketches, storytelling and prototyping as a means of analysis and design. By soliciting user feedback at each stage, the author and developer was able to quickly determine if potential users were able to understand interface objects. Norman & Draper [13] notes that user cantered design places the user at the centre of the design process, from the initial analysis of user requirements to testing and evaluation. Beaudouin-Lafon and Mackay [2] add that prototypes support this goal by allowing users to see and experience the final system long before it is built. They also note that because prototypes are concrete and detailed, it allows for the exploration of real-world scenarios, and users can evaluate them, and they can be compared to directly with existing systems and designers can learn about the context of use and work practices of end users.

10.1.2 Analysis and Synthesis

The chosen methodology required a need to shift constantly between two kinds of design activity, *analysis* and *synthesis* [18]. During analysis, the design was tested to determine whether it is meeting targets for usability and during synthesis the design was shaped by drawing on fresh ideas borne from user feedback and solutions to similar problems that have been worked in the past through observation of existing systems and review of literature on existing systems.

10.1.3 Design Steps

To design the interface for the system and for game I chose the storyboard and paper prototyping approach. It involves and process of sketches, storytelling and prototyping as a means of analysis and design. At each stage the developer is able to quickly determine if potential users were able to understand interface object. The techniques for interface design are described below with the rationale for use. In order to produce user friendly and usable interface I perform the following steps:

1. **Storyboards**

Before prototyping began, storyboards are used to capture more of the scope and flow of the design proposal [Snyder]. Storyboards are much like paper prototypes but broader in scope and not generally intended for input from the users. These documents become a record for all ideas and assumptions made during the designing process.

The storyboards for the system were done on the paper and attempt to capture the issue that need to be addressed, assumption, and observation of existing systems. The storyboards are rich source of information for designing any system for human users because it encourages good interface architecture.

2. **Paper Prototypes**

Following the storyboarding exercise where the domain was explored, rough paper sketches of several alternatives were created. In some cases, none of the alternatives sketched fully met the needs of the site and game, and so a lot of quick sketches were done utilising user feedback and ideas borne from existing similar systems. The paper prototypes of the system were sketched and attached in appendices (**Appendix C and D**).

10.2 Interface Structure

The Figure below is the main structure diagram for client program of JDFS. It shows how all the screens are related and how the user can move from one to another. The user can login or register to

the system. After login user is redirected to the client window where the actual interface is presented. The Interface Structure diagram is presented below (**Figure 26**).

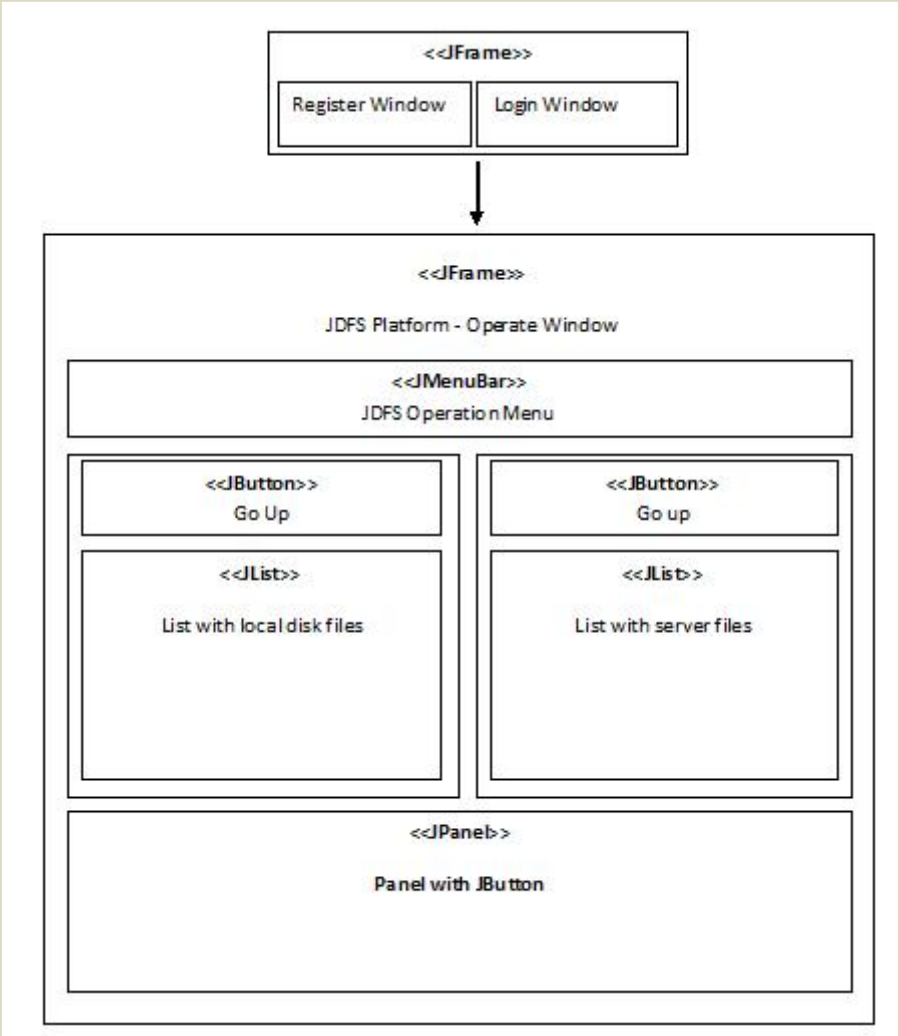


Figure 21: The interface structure diagram for client system

Chapter 11: Implementation

This is probably the most consuming phase in terms of time. This is where the system is actually built and installed. Certain crucial part of the code implemented will also be demonstrated.

11.1 Implementing the system

The system implementation, meaning that the client and server parts were implemented based on design set out in the previous sections. The client and server were written in Java. The reasons for choosing Java for implementation were described in Chapter 5. So, this chapter describes each classes used to build the overall system.

11.1.1 Server Implementation

The functionality of each server's classes is described briefly with the main functionalities.

- **ConnectionThread class**

This class is responsible for connection between the client and server, namely it deals with request and responses received. The most important public function is run() which starts the client section running. Always the client send request first to the server and the server is obliged to response to particular client's request. The run() function was implemented as a set of else if statements in order to enable the client and server run concurrently.

- **DistAccountList class**

This class deals with users accounts by storing users as object. This class contains one method:

- **AddUser() -** which adds the user to the list**,** so the user can use the system**.** The system produce a file called ServerUsers.xml where we can see all registered users. The part of the xml file is presented below:

```
<?xml version="1.0" encoding="UTF-
8"?><ServerList><ServerInfo><Name>java</Name><Ip>localhost</Ip><Port>1234</
Port></ServerInfo><ServerInfo><Name>java</Name><Ip>localhost</Ip><Port>84</
Port></ServerInfo></ServerList>
```

- **DistAccountManager class**

This class deals with user's accounts. The user's login and password is stored on the local machine in file called user.dat. The DistAccountManager class manages this list by adding new users during registration,

- **DistServer class**

This class is responsible for operations that the system can perform on files such as: rename, delete, copy, paste. The most important methods are:

- **Run()** – which starts the section running as a separate thread because the server has to listen on the specified port all the time.
- **LoadFile()** – method loads files from servers and enabling all users to see the same context of files.
- **DeleteFromFileList()** – method enables deleting files from server list but only by authorized users.
- **RenameFile()** – method enables renaming files from server list but also only by authorized users.
- **AddToFileList()** – method enables to add files to the server's file list and in the same time method is responsible for updating files on the servers.

- **DistServerFileListPartEntry class**

This class represents the name of the file. Also the file is divided into several pieces and so this class is used to identify on which server every piece is stored.

- **DistServerFileObject class**

This class represents a file or folder on local disk and server. This class is mainly responsible for adding files to the local disk or server file list.

- **ServerConfing class**

This class is the main class in order to run the server and allow communication. This class creates a simple interface in order to enter the port on which the software can communicate with client.

- **ServerLog class**

This class represents the JTextArea dialog for writing the server logs. The class is not responsible for generating the required logs but only for adding logs to the JTextArea dialog. However, the class can add the following code:

```
Message: Premature end of file.
[Ljava.lang.StackTraceElement;@1d381d2
2011/07/17 21:14:43 :::::: Exception during list reading   xmlaParseError
at [row,col]:[1,1]
```

This error was generated during system testing when the software did not support my computer architecture.

- **ServerWindow class**

This class represents the server window. The server window has a very simple interface and is used only to writing the actual server's logs during various operations.

11.1.2 Client Implementation

The functionality of each client's classes is described briefly with the main functionalities.

- **ClientConnectionThread class**

This class enables connection between the client and server. The main available method are described below:

- **Open()** – method enabling the client to open the connection on specified port in order to allow communication with the server.
- **Close()** – method enabling the client to close the connection with the server.
- **SendAsynch()** – method enabling passing the message from sender to receiver without waiting for the receiver to be ready
- **Send()** – method enabling synchronous message sending when the sender and receiver has to wait for each other to transfer the message.
- **Receive() –** method enabling synchronous message receiving when the sender and receiver has to wait for each other to transfer the message.

- **BrowseList class**

This class enables the system to display the folders and files as a list. This class is also responsible for refreshing the list with files.

- **BrowsePanel class**

This class represents the panel with files list on local disk and server.

- **Callback class**

The interface callback provide mechanism which is required for precise work during asynchronous message passing between the server and client.

```
public interface Callback
{
      public void Call(CommunicationFrame response);
}
```

Figure 22: The code for Callback class.

- **DistClient class**

This class represents the client object. Mainly the class provides control mechanism so it checks when the user has sufficient rights to perform the require operations.

- **DistClientLoginWindow class**

The main class enabling the client side to run and displaying the interface for client's login and password details. This interface also enables users to perform registration in order to get access to the system.

- **DistClientWindow class**

This class enables the user to see the main client's interface. This interface is displayed to user after providing correct login details (unique for each user).

The interface enables user to see the proper system namely, the left side consist the files available on the user local disk, and the right side consist files that are ready uploaded to the server by different users. The interface provides all system's features in order to maintain files rights (read, write, rename, delete, copy, paste).

The client interface also allows the use of quit or exit to end the program and it also gives a help screen detailing the syntax of all of the commands when the user choose help from menu at the top of the client's window.

- **DistInfoDialog class**

This class enables users to view the help context. It extends the JDialog , so dialog can be run as separate window.

- **DistServerAddWindow class**

This class is responsible for adding more servers to the systems. This is system feature allowing users to add more servers.

- **DistServerListManager class**

This class is responsible for managing the list with all servers. The main functions are:

- AddServerInfo() – method responsible for adding new servers if the server is not available on the current list.
- RemoveServerInfo – method responsible for deleting the server from list.

- **DistServerSettingsWindow class**

This class represents the list of servers and mainly is responsible for managing this list. The main method of this class is LoadServerList() and the code is showed below:

- **ServerInfo class**

This class enables the client to store the server's IP and server name.

- **SettingManager class**

This class is responsible for managing the server's list.

- **Swap class**

This class represents the object for file storage because the file has to be stored and updated all the time.

- **Transferable File Object class**

This class enables transferring object between the client and server and vice versa during the drag and drop operations. The most important public function is **getTransferData().** This method allows the transferring data, and also allows catching exceptions (throws UnsuportedFlavorException) during the transferring operations.

11.2 Software Prototype

The software interface prototypes were then built based on the best-fit sketch identified.

- **Software interface prototype for Client:**

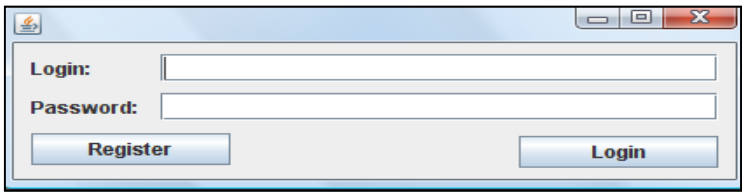


Figure 23: Client interface prototype - login window

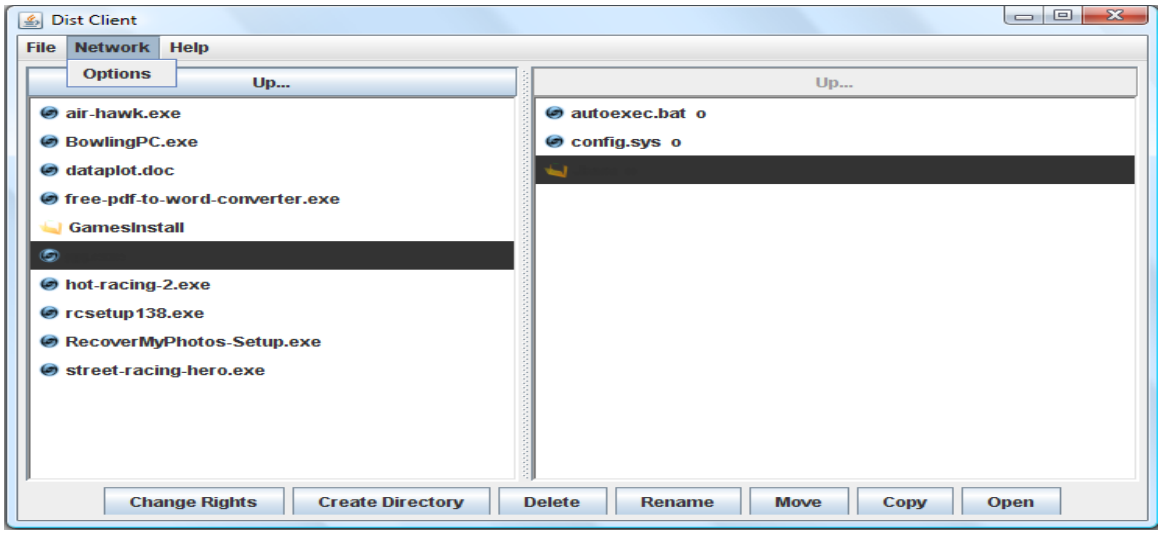


Figure 24: Client interface prototype – main system window

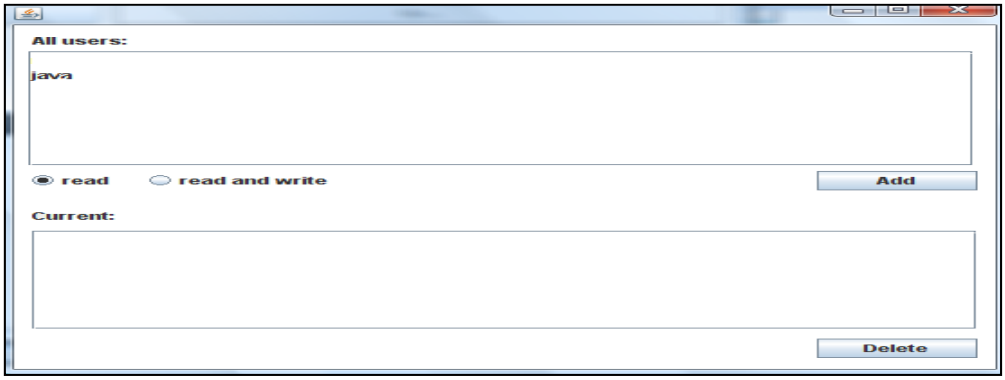


Figure 25: Client interface prototype - users can view their rights and change them

- **Software interface prototype for Server:**

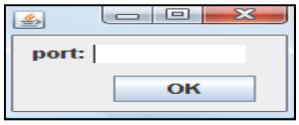


Figure 26: Server interface prototype - window for port number

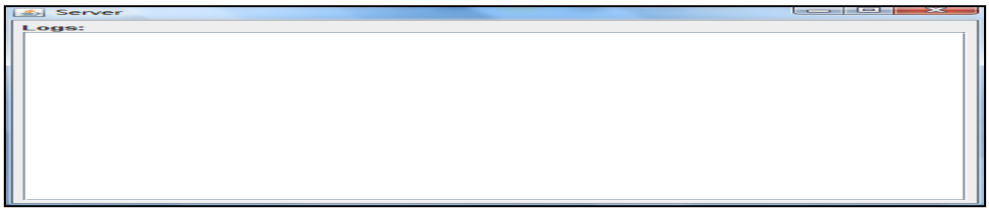


Figure 27: Server interface prototype - main server window where server logs will be displayed

This chapter describes the system features and shows how this system differs from other similar systems that are available in market.

12.1 System Features

The system has lots of feature and these features make the system different from others. They are as follow:

There are some features of my system and these features are related to Java:

a) System built in Java

- Java stands for programming language and for platform as well. It allows the programs written in Java to run on every machine such as: Windows, Linux or Mac.
- Java stands for powerful programming language allowing programmers to experiment with multi-threads. Java provides tools for developing networking applications and as well allows catching exceptions in sufficient way. It gives my systems set of advantages because Java make sure that my system will operate without errors.

a) Easy Interface

The good designed DFS should have easy interface. The JDFS has well designed interface because the user evaluation gave positive opinions.

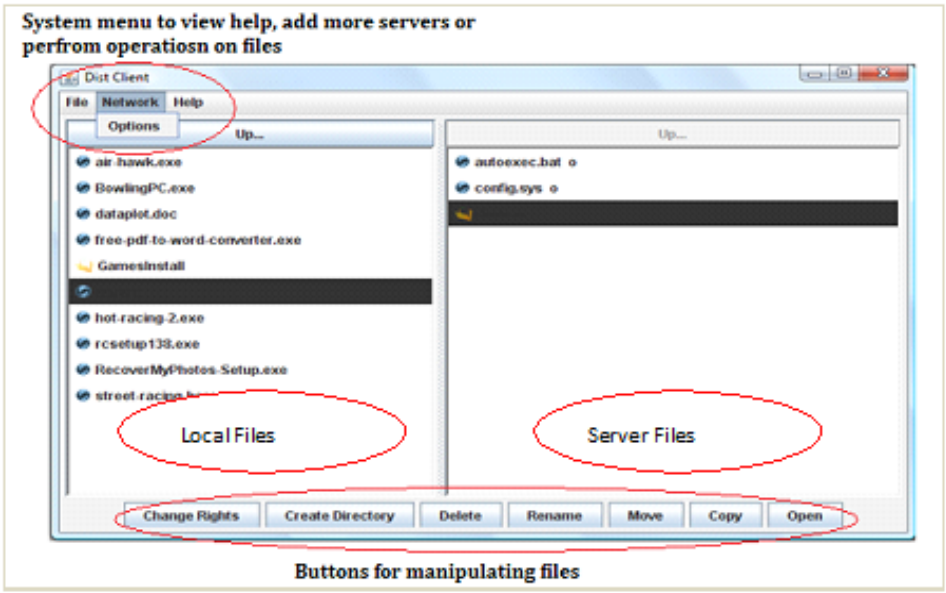


Figure 28: System interface

b) Sufficient File Replication

Let's consider the first scenario when we want to calculate the availability of replicas. First we have t use the formula showed below.

$$\rho_{prep} = 1 - (1 - \rho)^k$$

Where: ρ = node availability, k = redundancy factor, ρ_{prep} = file availability

Example: p = 0.25, k = 5, ρ_{prep} = 0.763 so it gave a value that is not enough, even if we increase the k factor is always gives a value that could be unrealistic.

I used the same approach that was used by Wuala - P2P online storage. Wuala uses erasure codes. Erasure codes means that we need to encode m fragments into n need any m out of n to reconstruct. How the availability of replica is now calculated? We need new formula as follow:

$$\boldsymbol{\rho_{ec} = \sum_{i=m}^{n} \binom{n}{i} p^i (1-\rho) n - 1}$$

Example ρ = 0.25, m = 100, n = 517, k = n/m = 5.17, ρec = 0.999 so it gave a value that is enough and as well is realistic in terms how many replicas we need to produce.

Software testing is the investigation of a software product intended to provide information about its quality. In other worlds, testing is an essential process and is concerned with the verification that the software is working properly. Based on the requirements in the document, many attempts are done to test every components or modules that the software is composed of. There are several ways to test a piece of software.

The software testing methodology proposed for this project involved incremental integration testing were the application was tested as new functionality was added. The software after that was finally tested by user (volunteer). It was split into three phases: Unit testing, System testing and Acceptance testing did by developer during the development process.

13.1 Unit Testing

Unit testing play important role in a software development process. According to [14] a unit is a smallest testable part of an application (classes in object oriented programming) and it explores a particular aspect of the behavior of the class.

There are two approaches to unit testing: black-box and white-box. Black-box is the most commonly used since each class represent as encapsulated object. White-box testing is based on the method specification with each class.

13.1.1 Black – box Testing

Black-box testing is an approach to testing where the tests are derived from the program or component specification. The system is a black box whose behavior can only be determined by studying its inputs and the related outputs: it can be called functional testing. Black-box testing strategy examines the observable behavior of software as evidenced by its outputs without reference to internal functions. It is also not necessary to understand everything that is invisible inside the system.

During testing all the system functionalities are examined. I am going to show a list of tables with testing results:

Functionality	Input	Action	Actual Output	Result
Login	Username and Password	Click ‘Log in’ button	Pass and entry the system interface	No error found
Registration	Username and Password	Click ‘Register’ button	Submit to the system and get confirmation message	No error found
Copy a file	Select a file to copy	Click ‘Copy’ button then click place when you want to paste a file	The file is placed in the selected place	No error found

Cut a file	None	Click 'Cut' button	None	No error found
Paste a file	Select a place to paste a file	Click 'Paste' button	The information is passed to the system and the information is presented on the screen	No error found
Open a folder	Select a folder	Double click on the folder	None	The system is able only to open txt and doc files
Create a folder	Select a place for new folder	Click 'Crete folder' button	Get the confirmation message from system	No error found
Read a file	None	Double click on the file	None	No error found
Write a file	None	Double click on the file	None	No error found
Rename a file	Select a file	Press the 'Rename' button	The name of the file is changed and presented to user	No error found
Delete a file	Select a file	Press 'Delete' button	Get the confirmation message from system	No error found
Exit JDFS	None	Press 'Exit' button	Get the message from system	No error found

Table 9: System testing result

Functionality	Input	Action	Actual Output	Result
After server crash user is still able to download file	Username and Password	Click copy and paste button	Pass and entry the system interface	No error found
The files are divided into several replica		Open Server file called Server.xml	The context of xml document is telling use where	No error found

			the files are	

Table 10: Server testing result

13.1.2 White – box testing

This type of testing is based on a small portion of the program that needs to be tested. This time, the full assembled code will be fully tested. The results are documented and need to be studied in greater details when the testing has been completed. The outcome of such study will reveal if the software, described in the requirements, behaves in a correct way. The report must state:

1) What was being tested?
2) Were the requirements were met?
3) Was the testing successful and was there any error revealed?

A system can be tested at several levels.

1) A routine can be implemented in the program to verify whether it is in a working condition. This is done by creating test data.
2) The system integration process begins when all the components are implemented, checked and put together to form the final system. When this process is completed, the whole system is checked again and again to make sure there is no 'bug' in the software. In this way, interface problems may be discovered, as well as other types of errors.

13.2 Usability Testing

In the user – oriented paradigm usability is a big issue. It is extremely important to keep usability in mind during the whole process of designing. The whole system should be efficient, easy and satisfying to use. The software application should be suitable for the target to best achieve their goals.

13.2.1 Usability Evaluation – results

The first few steps to prepare usability testing are identifying the purpose of the test, defining the user group, specifying problem statements, identifying user task, and specifying the performance data to be collected. [16]

Then the real evaluation process began with users answering a questionnaire and using the software. The questionnaire consists of simple instructions about the devices and 9 questions regarding the overall or independent functions about the system. Users were asked to rate the system from 1 to 5 (higher score is preferable) and more preferably to give feedback and suggestions (**Appendix K**).

The software application was tested on 10 students.

It gives a positive aspect about the whole System interface because most of the users rate the system very high (**Figure 34**). Also, the system function evaluations (**Figure 35**) were rated high, so it means that most of the user found this system easy to use.

Together with the satisfactory overall rating, positive comments were received as the user's feedback. These comments include:

User 1 (Anonymous): "*The system is easy to use, and provides lots of interesting features. I was quite surprised how easily the files can be shared between different users".*

User 2 (Anonymous): "*The system presents well and has a strong function in general".*

However, critics and constructive suggestions were collected from the user evaluation. The feedback includes *"Need several minutes to learn at first time.", "The system should have better help for new users".*

Furthermore, some software design bugs were also detected in the usability test. Some user gave their opinions regarding the more favorable and convenient system function.

By analyzing the test result, some improvements should be made to enhance the usability. In fact, the system was tested and re-designed many times to maximize its function.

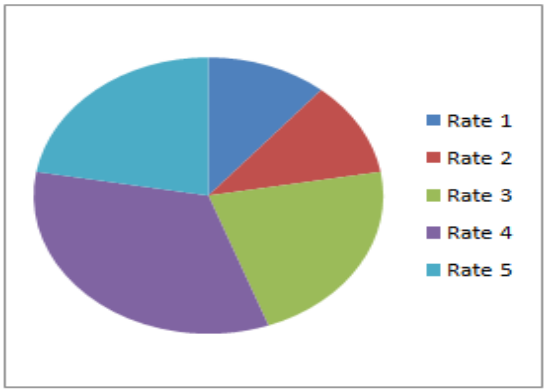


Figure 29: System Interface evaluation

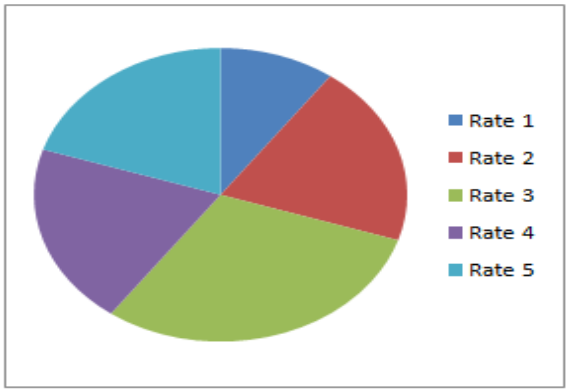


Figure 35: System functions evaluation

Chapter 14: Conclusion

14.1 Project Conclusion

The development of the Java Distributed File System is successfully completed. I gathered and implemented all major requirements for this software. Due to many unexpected problems arises, a lot of intended functionality had to be cut from the original system observations. Despite of many changes made, a working system was finally achieved, which satisfied the original objectives.

Many problems had already been considered during the development phases, meaning problems that occurred were dealt quickly. The project is certainly an accomplishment for me after 4 months of hard work. I am quite familiar with server/client model, file replications techniques and advanced programming in Java. In addition, I have gained knowledge with project management and system engineering. I have also learned about every aspect of design. The project has been the most rewarding and beneficial undertaking of my student career.

The JDFS file operational rule seriously follows the traditional file operational rule, such as that in Windows and Linux. The transmitting file's information is encapsulated into a message, the remote machine can prejudge whether the files whose names contained in the message exists and has the appropriate permission and status, and then do relevant operation. In this way, the JDFS has a high security on file communication. The JDFS is also capable of removing the temporary files in order to save disk space and keep away rubbish file. This always happens when the local user reads and edits remote file, there is a temporary file left in Temp folder during the 'read' or 'write' operation. As long as the 'read'/'write' operation is done, the temporary file will be removed. It could not be mentioned that the JDFS has the strong ability l shared folder, if the files confliction of ct user interface is f users to accept it.

14.2 Recommendation for Future Work

My plan for the JDFS is simple: To make the system more s. Of course, to accomplish this goal, much work remains to be done. Other modifications to the JDFS design include:

- The JDFS was initially designed to operate in one small workgroup, but JDFS should also be suitable for multiple workgroups and networks.
- This JDFS might save the space of network by certain compression technique for multiple files transmission.
- This JDFS should make it possible of multiple users' responds to the request from aspecific user in the network, if they have the ability to meet the request. The JDFS will find the nearest and suitable machine to serve the user.
- This JDFS need make the copy operation more powerful, which it can not only copy remote files and folders in local shared folder but also in local unshared folder.
- This JDFS should use Data Mining techniques of AI to capture the shared files by varied permission in a huge shared file warehouse.

References

[1] A. Silberschatz, P.B. Galvin and G. Gagne, "Operating System Concepts ", Sixth Edition, Chapter 16 Distributed File System, 2003, pp. 537-578, 585-588.

[2] Beaudouin-Beaudouin-Lafon, M. and Mackay,W. (2003) Prototyping Tools And Techniques. Jacko, J. A. and Sears, A. (eds.) *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*. Lawrence Erlbaum Associates, Mahwah, NJ, USA, 1006-1031.

[3] Elizer Levy and Abraham Silberschatz, Distributed File Systems: Concepts and Examples, Department of Computer Science, University of Texas at Austin, Austin, Texas 78712-1188 accessed on 07/06/2011 at http://citeseer.ist.psu.edu/viewdoc/download;jsessionid=559C858A11331DCB388A61811F3A816A?doi=10.1.1.109.2577&rep=rep1&type=pdf.

[4] G. Coulouris, J. Dollimore, Tim Kindberg, Distributed Systems Concepts and Design, Pearson Education Limited, 2001, pp. 309-310.

[5] H.M. Deitel, P.J. Deitel "Java How to Program" Fifth Edition, 2003.

[6] J. Magee and J Kramer, "Concurrency State Models & Java Programs", Chapter 10 Message Passing, 1999, pp. 205-221.

[7] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, The Hadoop Distributed File System, Yahoo, Sunnyvale, California USA, accessed on 26/06/2011 at http://docs.google.com/viewer?a=v&q=cache:nP6tMBFBgJEJ:storageconference.org/2010/Papers/MSST/Shvachko.pdf+hadoop+distributed+file+system&hl=en&pid=bl&srcid=ADGEEShYiI1jd8_gQUUNXjOUPZCTrVfrAoXeTOk3b1NMUO4uiUTxQFWcDj47VlBCddo5MfoT4AydetLA9qisnj2uBN_YdG-TfGPmB9unuvOaM93pfgXQhIp6_hERrRPzMxlgpFWD5Q2L&sig=AHIEtbQFr-Md2dMI8WQ7X6My9eitpUt6Xw.

[8] Leszek Maciaszek, *Requirements Analysis and System Design*, Reading, Massachusetts: Addison Wesley, 2nd Edition, 2005.

[9] Morris Sloman and Jeff Krammer, "Distributed Systems and Computer Networks", Prentice-Hal international, 1987, pp. 1-3.

[10] Marc Mancini, Time Management - 24 techniques to make each minute count at work., Mc Graw Hill, 2007, pp. 1-15.

[11] Markus Aleksy, Axel Korthaus, Martin Schader, Implementing Distributed System with Java and Corba, Springer, University of Mannheim, pp. 6 – 55.

[12] Mark Burgess, Principles of Network and System Administration, Oslo University College, Norway, John Wiley & Sons, Second Edition, pp. 20 – 45.

[13] Norman, D. A., & Draper, S. W. (Eds.) (1986). *User centered system design: New perspectives on human-computer interaction***.** Hillsdale, NJ: Lawrence Erlbaum Associates.

[14] Newman, W., Lamming, M (2005) *Interactive System Design*, Addison- Wesley, Reading, MA

[15] Smita Hegde, "Replication in Distributed File Systems", Department of Computer Science, University of Texas at Arlington, accesses on 03/06/2011 at http://crystal.uta.edu/~kumar/cse6306/papers/Smita_RepDFS.pdf.

[16] Sun Network File System (NFS), accessed on 01/07/2011 at: http://pages.cs.wisc.edu/~remzi/OSFEP/dist-nfs.pdf.

[17] Tom White, Hadoop The Definite Guide, O'Reilly, Yahoo Press, 2nd edition, pp. 31 – 55.

[18] Webobedia, "What is a protocol?", accesses on 29/05/2011 at http://www.webobedia.com/TERM/P/protocol.html.

Appendices

Appendix A: Critical Review and Reflection

Motivation

I had very strong motivation during the project. The project involved lots of programming in Java, and I am very keen for programming, because Java is one of my favourite programming languages. I chose this project because I wanted to do something interesting and complex as well. I know that DFS is itself very complex and interesting because it is still subject to research for improvements.

I believe that I mastered my programming skill, because this project cost me lost of coding. I produced nearly 7,000 lines of codes and the system met most of the requirements.

The most interesting aspect of this project

The most interesting aspect of this project was that I spent plenty of time on improving my programming skills. As I mentioned before I wrote nearly 7,000 lines of codes (including the brackets as line) and it was very challenging. It helped me to understand complex structure of DFS, and I believe that it will benefit in my future career as a Java programmer.

Problem encountered

There were many problems encountered and solved during the project. Some of the problems are listed below:

- **During the Planning stage**
  During the planning stage, I spent a lot of time on getting and gathering ideas by searching of articles from websites and reading books. It was a painful and time consuming experience. There were a lot of problems according to software modeling technologies which should be used, the client and server model had to understand in deeper (ports, sockets, tunneling) and many more.
- **During the Design stage**
  After getting the understanding how the system should look plenty diagrams were drawn. The diagrams were changed many times to get the best model for implementation. It was also painful experience because modeling is not as easy as it seems to be.
- **During the Implementation stage**

During this stage the code has been produced. At the end all the codes were working without any errors or warnings. The implementation stage took me about 50% of time which has been allocated to this project. I found really difficult to implement all the system features which I planned to do. After searching and comparing the codes form several books, finally I got all the necessary ideas. But this stage was the most time consuming and the most creative stage in whole project live.

Skill Review

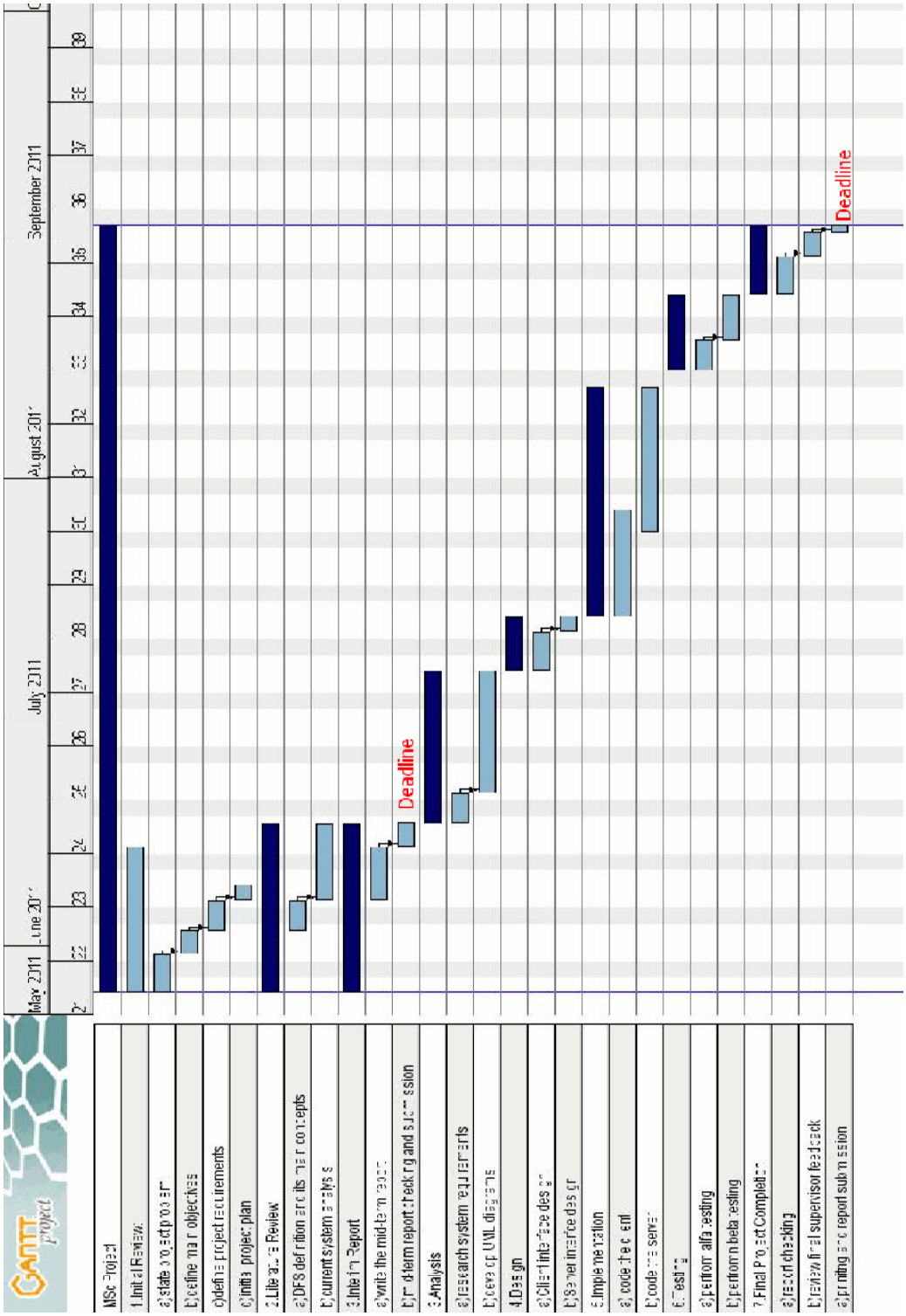
I have achieved the skills of getting and gathering ideas by searching articles and journals from websites, reading of books and project guidance. I have also improved the skills in networking programming and testing. Programming skills were improved by writing complex working system for users to perform some set of file's operations. This system required to perform testing to meet all the objectives for the system. Skills of assessing and evaluating project progress through setting of targets and deadlines are also achieved.

Reflection

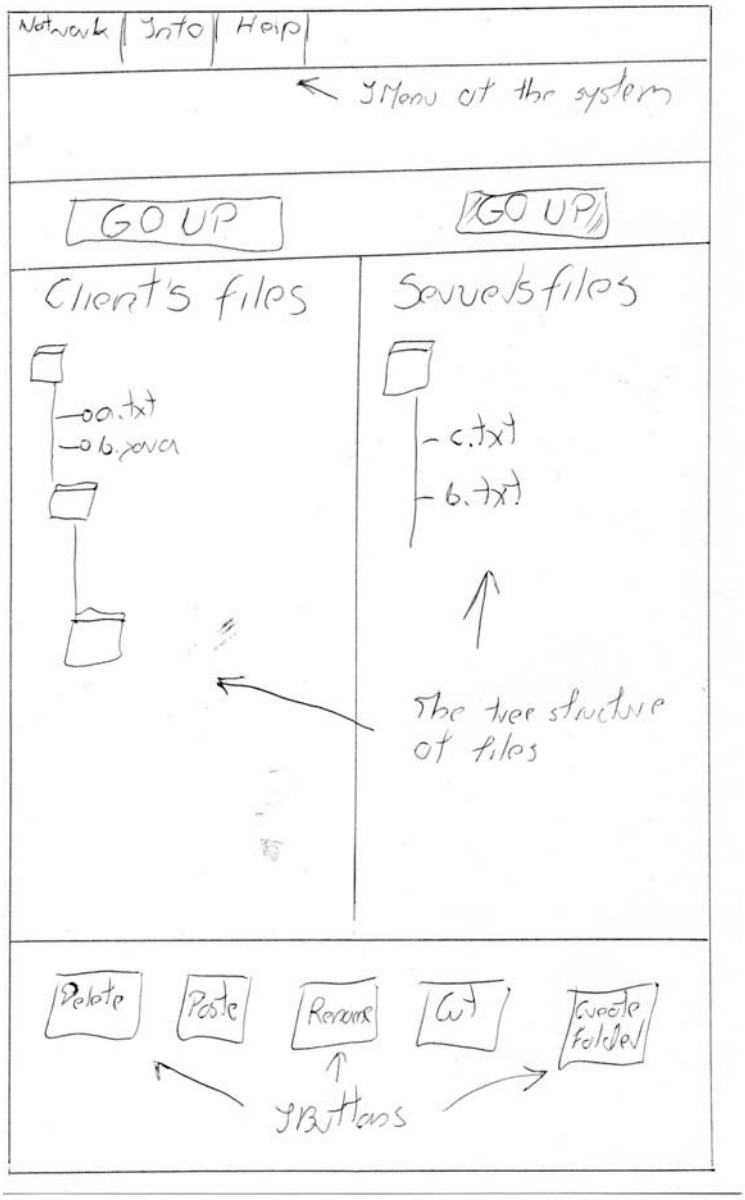
I strongly believe that I have managed my time reasonably well and have achieved an understanding of many aspects of project management. In particular, I had encountered the difficulty of estimating the time, which is required to complete each task assigned. The project required to build working system according to software principles. Some of the aspects from the Software Development Live Cycles were not touched upon in regular classes. At the beginning of the project I had little understanding on UML diagrams, so I spent plenty time in library to search information about software modeling. At the end of the project software has been produced according to the UML diagrams. The code has been produced and compiled without errors and warnings. At the beginning of project I try to write the software without modeling first. It was wrong idea because the system could fail in time constraint and probably will not meet user expectation. I think I have built a solid foundation for further improvement. I also learnt about researching during the project.

Overall I am sure that the time spent has proved to be valuable educational experience.

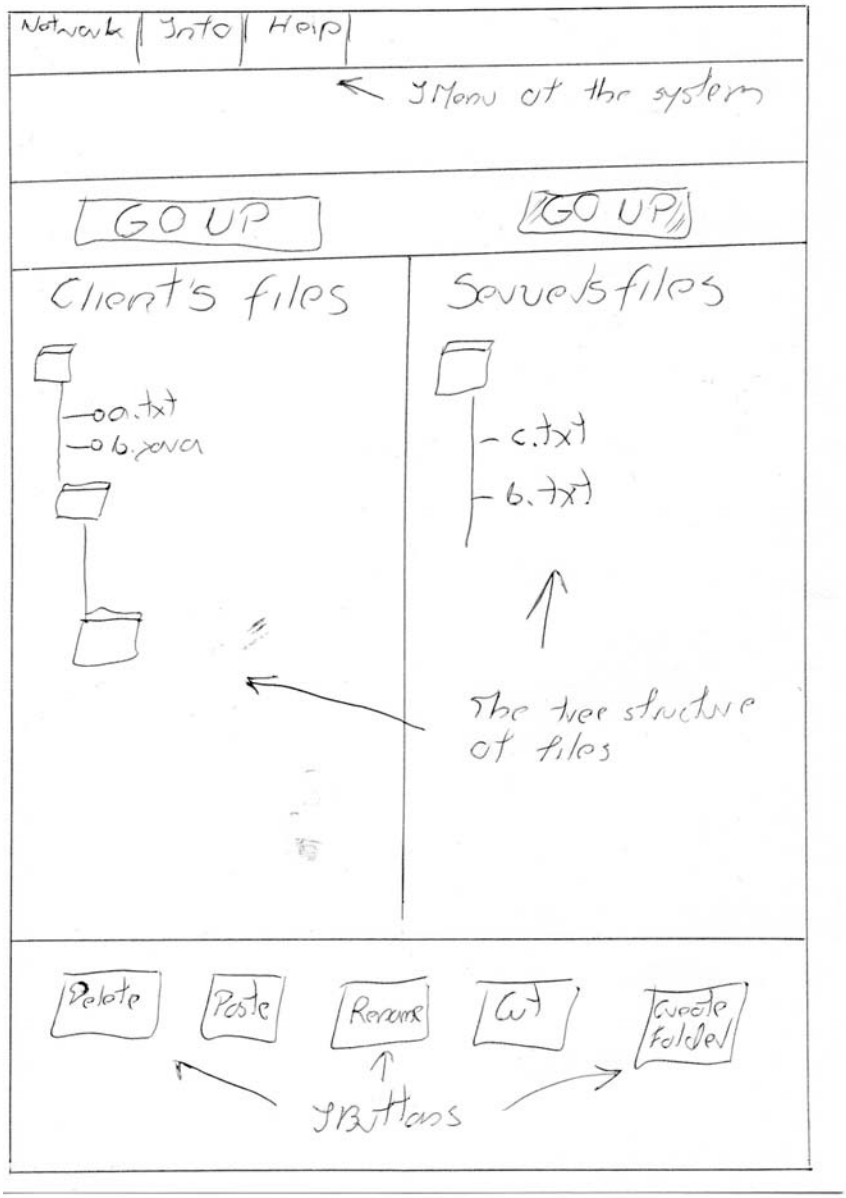
Appendix B: Project Plan



Appendix C: Interface Paper prototype – Client



Appendix D: Interface Paper prototype Server



Appendix E: Connection Sequence Diagram

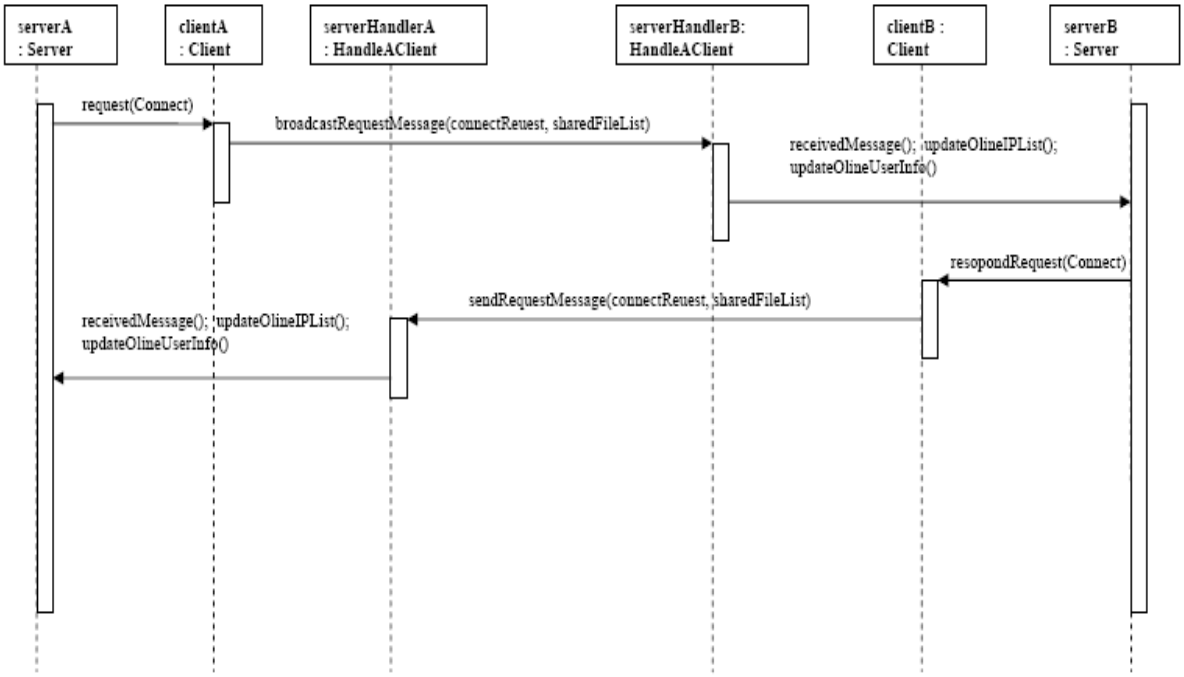


Figure 31: Sequence Diagram for Connection

Description:

- The sequence starts when the local user A begin to connect to network, that is, clicking 'Connect' button.
- Local server A sends 'connect' request by construct object client A of Client Class.
- Client A searches for remote online users from IP list and send local shared file list with 'connect' request to other online users.
- Remote client B's sever handler gets the shared file list and sends it to server.
- Server B stores machine A's information and update its remote-online-user-IP list
- Remote server B respond to connection by sending 'connect' request and its shared file list to local client A's server handler.
- Local client A's server handler accepts the request and sends machine B's shared file to machine A's server.
- Server A stores machine B's information and update its remote-online-user-IP list.

Appendix F: Disconnection Sequence Diagram

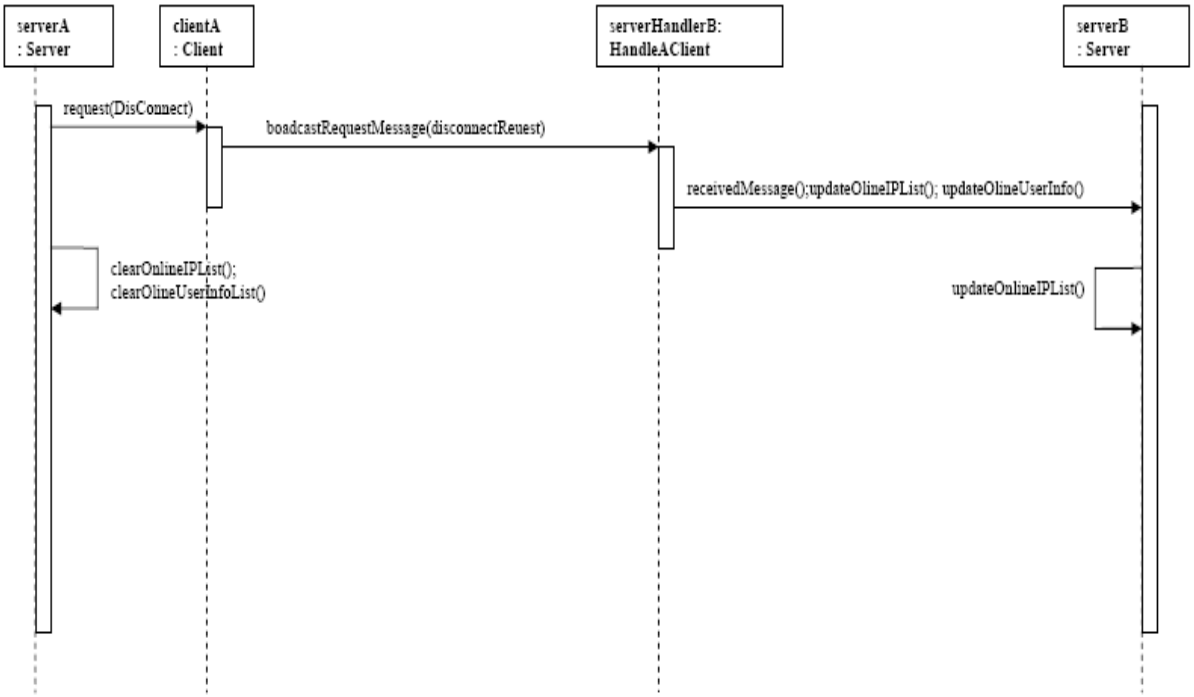


Figure 32: Sequence Diagram for Disconnection

Description:

- The sequence starts when local user A attempt to disconnect from network.
- Local server A broadcast 'disconnect' request to remote online users on its remote online-user-IP list by creating an object of class Client.
- Local server A clears up its remote-online-user-message list and update its remote-online-user-IP list.
- Remote client B's sever handler gets the request and send to its server.
- Server B remove machine A's information from remote-online-user-message list and machine A's IP from its remote-online-user-IP list.

Appendix G: Delete/Paste/Rename/Cut Sequence Diagram

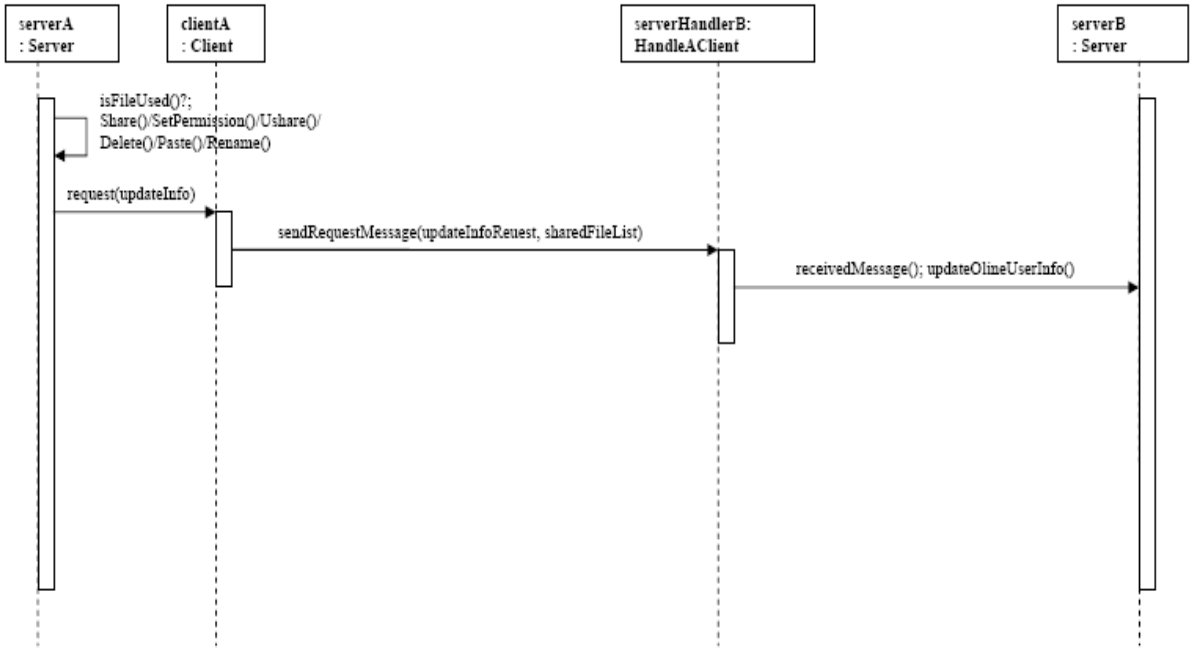


Figure 33: Sequence Diagram for Delete/Rename/Paste/Cut

Description:

- The sequence starts when local user A share/unshared/delete/paste/rename/set permission to a local shared file while its machine is online.
- Local server A sends 'update info' request by construct object client A of Client Class.
- Client A searches for remote online users from it remote-online-user-IP list and send local shared file list with 'update info' request to online users.
- Remote client B's sever handler gets the shared file list and sends it to server.
- Server B updates machine A's information.

Appendix H: Copy/Paste Sequence Diagram

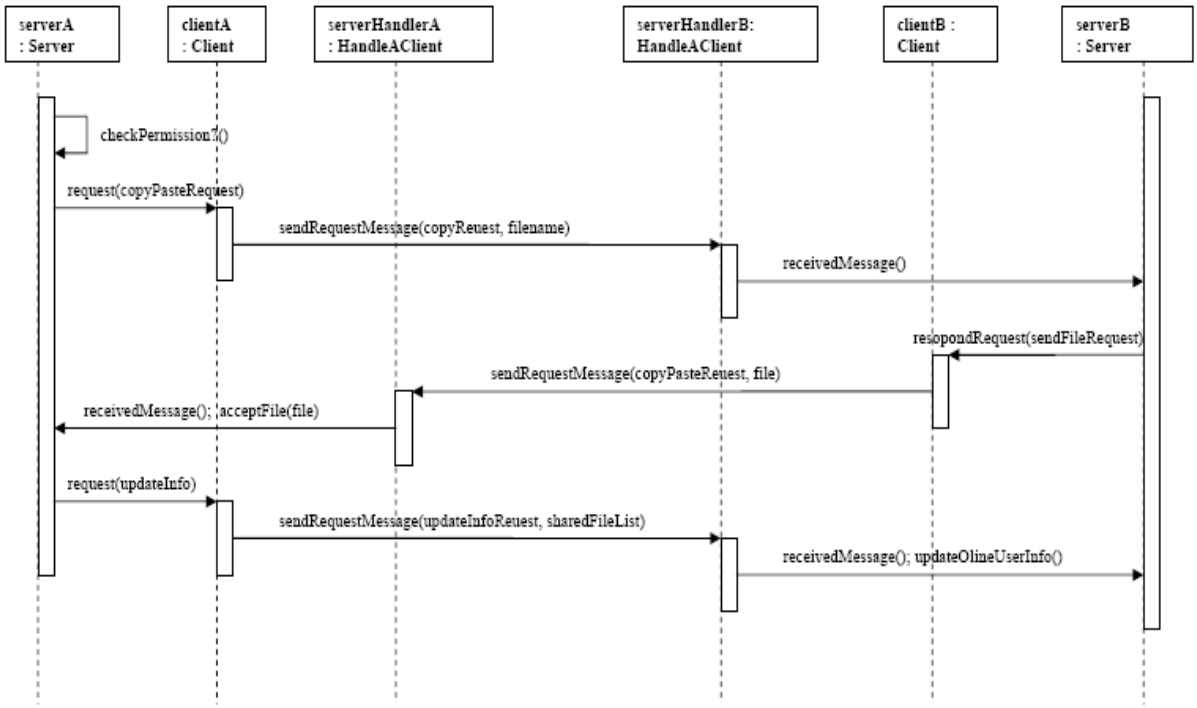


Figure 34: Sequence Diagram for Copy/Paste

Description:

- The sequence starts when local user A begin to copy a remote file from a remote online client B on network, i.e., selecting remote file and clicking 'Copy' button and then 'Paste' button.
- Local server A checks the permission of remote file
- If the file has 'copy' permission, local server A will send 'copy/paste' request and 'filename' to client B by construct object client A of Client Class.
- Remote client B's server handler receives message and sends it to server B.
- Local client A start to send its new user info to other remote online clients.
- Local server A sends 'update info' request to remote online users on its remote-online-user-IP list by construct object client A of Client Class.
- Client A searches for remote online users from it remote-online-user-IP list and send the local shared file list with 'update info' request to online users.
- Remote client B's sever handler gets the shared file list and sends it to server.
- Server B updates machine A's information.

Appendix I: Read a file Sequence Diagram

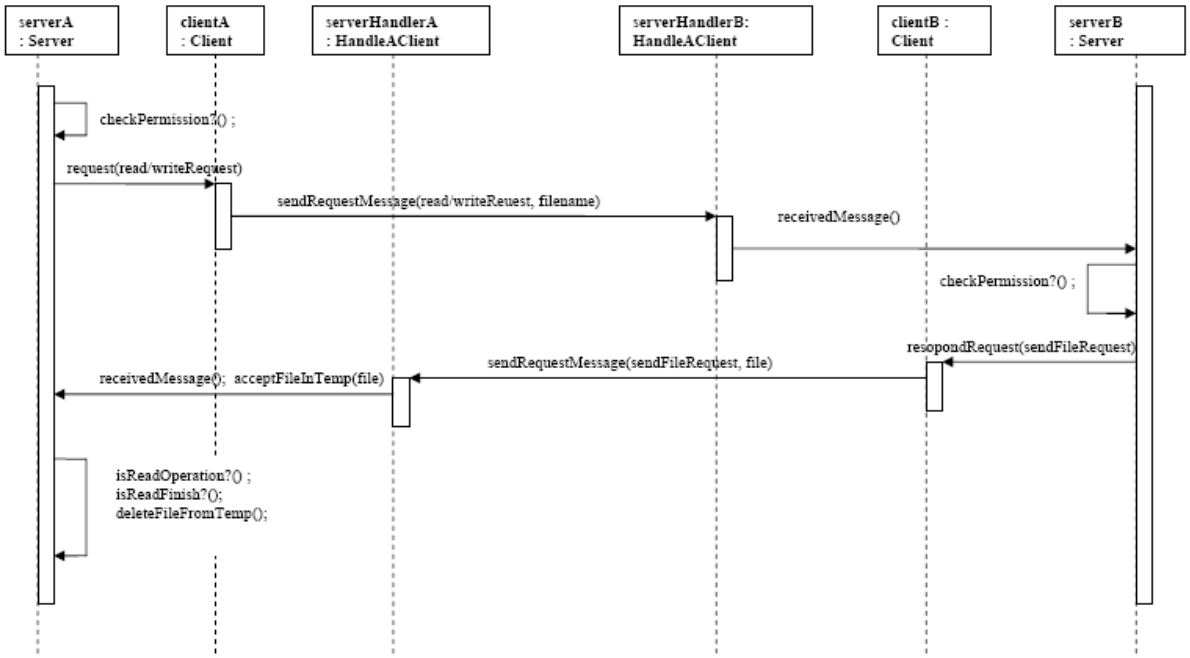


Figure 35: Sequence diagram for Read file

Description:

- The sequence starts when local user A begin to read/write a remote file from a remote online client B on network, i.e., selecting remote file and clicking 'Read' button .
- Local server A checks the permission of remote file
- If the file has 'read' permission, local server A will send 'read' request and 'filename' to client B by construct object client A of Client Class.
- Remote client B's server handler receives message and sends it to server B.
- Server B transfers the file and 'send file' request to local client A.
- Machine A's server handler received the file and request, and then sends to its server A.
- Server A keeps the file in its temporary folder.
- If it is 'read' operation, user A can read the file. As the 'read' operation is finished, the file will be removed from temporary folder.

Appendix J: Write a file Sequence Diagram

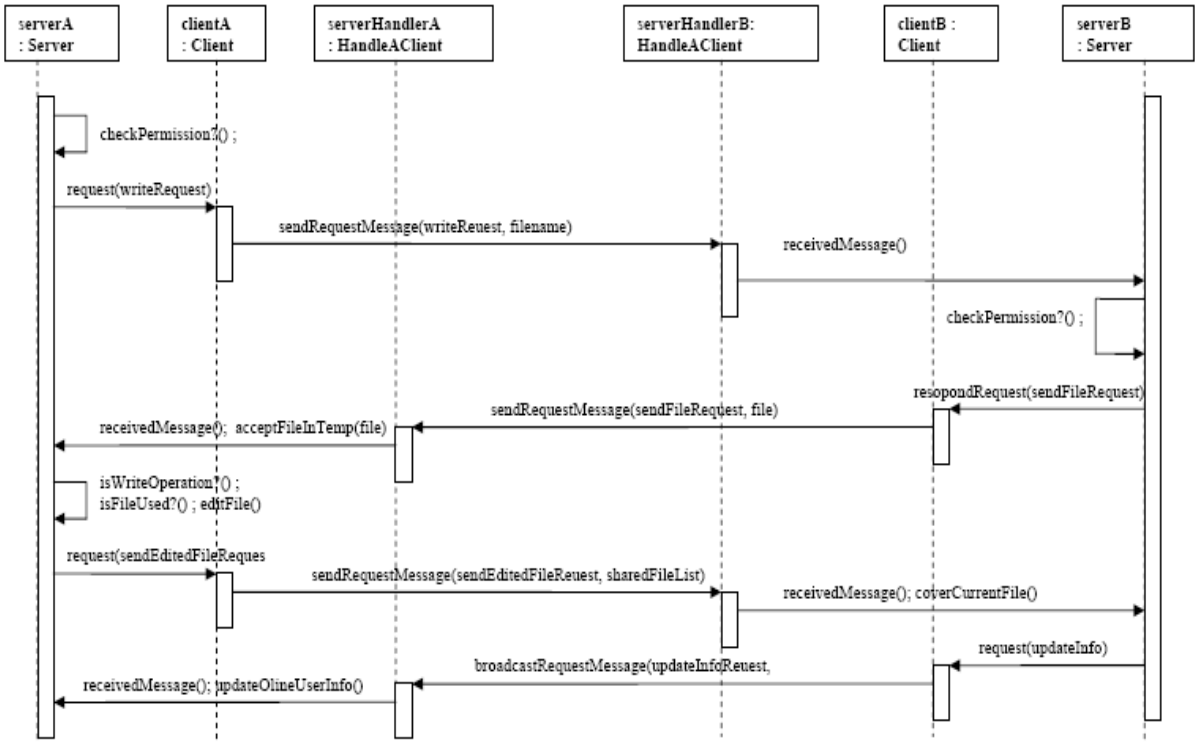


Figure 36: Sequence diagram for Write a file

Description:

- The sequence starts when local user A begin to write a remote file from a remote online client B on network, i.e., selecting remote file and clicking 'Write' button.
- Local server A checks the permission of remote file.
- If the file has write permission local server A will send write request and filename to client B by constructing object A of Client Class.
- If the file has 'write' permission, local server A will send write request and 'filename' to client B by construct object client A of Client Class.
- Remote client B's server handler receives message and sends it to server B
- Server B checks the permission of the file.
- Server B transfers the file and 'send file' request to local client A.
- Server A keeps the file in its temporary folder.
- If it is 'write' operation server A will check the permission and the status of file.
- If the file is allowed to edit now, user A can write the file, and then server send it back to its owner user B by creating object A of class Client.
- Server A transfers the file and 'send file' request to remote client B.

Appendix K: Questionnaire

User Evaluation Form

I am a postgraduate student looking to develop a Java Distributed File System prototype.

It would be of great assistance if you would complete the following questionnaire. Thank you.

Please mark the appropriate box with a cross[**x**].

Ratings: **1 = Excellent 2 = Good 3 = Average 4 = Poor**

PERSONAL DETAILS

Sex: Male [ ] Female [ ]

Age: 16-24 [ ] 25-34 [ ] 35-44 [ ] 45-54 [ ] 55-64 [ ] 65+ [ ]

Knowledge / Experience of:

	NONE	AVERAGE	EXPERT
Distributed Systems	[ ]	[ ]	[ ]
Search Engines	[ ]	[ ]	[ ]
Operating System	[ ]	[ ]	[ ]

Question 1

Please indicate your overall reaction to the system?

[1] [2] [3] [4] [5]

Question 2

What is your opinion of the following statements? (Please cross/tick the box that most closely matches your opinion.)

	Strongly	**Agree**	**Disagree**
a. I think I could use this system more frequently			
b. I think the system is intuitive and easy to use			
c. I feel very confident in using this system			
d. I found the system awkward			

Question 3

How important do you rate the following potential benefits? (Please cross/tick the box that most closely matches your opinion.).

	Very Important		Not At All
a. Easy file sharing with other users			
b. System allow communication with other users			
c. Access to huge pool of files and information			
d. System security is on very high level			

Question 4

Please describe which features of this system are most useful for you?

Question 5

Are there any other changes you would like to see in this system?

[Yes] [No]

If so what?

Thank you very much for you assistance.