

Distributed Information System (DIS)
RMI and Java 1.2 Implementation

by

Vicky Shiv

B. Tech. (Bachelor of Technology Computer Science)

Harcourt Butler Technological Institute, 1995, India

Thesis

submitted in partial fulfillment of the requirements for
the Degree of Master of Science (Computer Science)

Acadia University

Spring Convocation, 2000

©by Vicky Shiv, 2000



National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*

Our file *Notre référence*

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-52001-3

Canada

Contents

CONTENTS	IV
LIST OF FIGURES	VII
ACKNOWLEDGMENTS	VIII
ABSTRACT	IX
CHAPTER 1	1
1 INTRODUCTION	1
1.1 OVERVIEW OF THE THESIS.....	4
CHAPTER 2	5
2 CLIENT/SERVER ARCHITECTURE	5
2.1 CLIENT/SERVER MODEL	6
2.1.1 Client	7
2.1.2 Service request	7
2.1.3 Client/server interaction	7
2.1.4 Server	8
2.1.5 Service instance.....	8
2.1.6 Service interface.....	8
2.1.7 Service reference	8
2.2 CLIENT/SERVER COMMUNICATION	9
2.3 CLIENT/SERVER RESPONSIBILITIES	10
2.3.1 Client Responsibilities:.....	10
2.3.2 Server Responsibilities:	10
2.3.3 Middleware Responsibilities:	10
2.4 TIERS	11
2.4.1 Two-tier Architecture.....	12
2.4.2 Three-tier architecture	14
2.5 EXPECTED BENEFITS OF CLIENT/SERVER COMPUTING.....	16
2.6 SUMMARY	17

3	DISTRIBUTED COMPUTING	19
3.1	ADVANTAGES OF DISTRIBUTED COMPUTING.....	20
3.2	TECHNOLOGIES OVERVIEW	21
3.2.1	DCOM - Distributed Component Object Model	23
3.2.2	CORBA - Common Object Request Broker Architecture.....	24
3.2.3	Java RMI - Remote Method Invocation	25
3.3	ADVANTAGES OF RMI.....	27
3.4	SUMMARY	29
 CHAPTER 4		31
4	INTRODUCTION OF DIS	31
4.1	PUSH AND PULL MODEL	32
4.2	MAIN FUNCTIONALITY	32
4.3	COMPONENT OF DIS.....	34
4.3.1	Local Information System (LIS)	34
4.3.1.1	File.....	34
4.3.1.2	Folders.....	35
4.3.1.2.1	Workspace Folder	35
4.3.1.2.2	Downloadspace folder.....	36
4.3.1.2.3	Connected workspace folder	36
4.3.2	DIS Name Server	37
4.4	OPERATIONS.....	38
4.4.1	Off-line operations	38
4.4.2	On-line operations	40
4.5	SECURITY MODEL	42
4.5.1	Name server Security model	42
4.5.2	LIS Security model.....	42
4.6	OBJECT PERSISTENCE	43
4.7	EXCEPTION HANDLING.....	43
 CHAPTER 5		44
5	IMPLEMENTATION OF DIS	44
5.1	ANALYSIS.....	44
5.1.1	Strategy	44
5.1.2	Analysis.....	45
5.2	DESIGN.....	47
5.2.1	Architecture.....	47
5.2.2	Model	48
5.3	IMPLEMENTATION.....	50
5.3.1	View Model.....	51
5.3.2	Data Model.....	54
5.3.3	Communication Model.....	56
5.3.3.1	Name Server	56
5.3.3.1.1	Remote Interface	56
5.3.3.1.2	Implementing the name server	57
5.3.3.1.3	Starting the name server	58
5.3.3.2	LIS Server.....	59
5.3.3.2.1	Remote Interface	59
5.3.3.2.2	Implementing the LIS Server	59
5.3.3.2.3	Starting the LIS Server	59
5.3.3.2.4	Connecting to the name server	61

5.3.3.2.5	Connecting to the other LIS	62
CHAPTER 6	63
6 APPLICATION OF DIS: DRPE	63
6.1	NAME SERVER.....	65
6.2	DRPEC	66
6.2.1	Workspace Tab.....	66
6.2.2	Download Space Tab	69
6.2.3	Name Server Tab.....	70
6.2.4	Security Management Tab	71
6.2.5	User Management Tab	72
6.3	SUMMARY	73
CHAPTER 7	74
7 CONCLUSIONS	74
7.1	CONCLUDING REMARKS	74
7.2	FUTURE WORKS	76
BIBLIOGRAPHY	78
APPENDIX A	81
DIS INSTALLATION GUIDE	81
I)	SOFTWAREW INSTALLATION	81
II)	RUNNING LIS ON-LINE	82
III)	RUNNING LIS OFF-LINE.....	84
II)	RUNNING NAME SERVER.....	84

List of Figures

Figure 2.1 – Client/server Model	6
Figure 2.2 – Two-tier Architecture.	12
Figure 2.3 – Three-tier architecture.....	14
Figure 3.1 – Distributed Architecture.....	21
Figure 4.1 – Sample On-line operations.....	40
Figure 5.1 – Model View Controller.....	45
Figure 5.2 – Architecture of DIS.....	47
Figure 5.3 – Model-Delegate Design Model.....	48
Figure 5.4 – Implementation Model of DIS.....	50
Figure 5.5 – UML of View Model.....	53
Figure 5.6 – UML of Data Model.....	54
Figure 5.7 – UML of Data Model.....	55
Figure 6.1 – Name Server.....	65
Figure 6.2 – DRPEC Workspace Tab.....	66
Figure 6.3 – DRPEC workspace (Exporting a file).....	67
Figure 6.4 – DRPEC File Preview Dialog.....	68
Figure 6.5 – DRPEC Download Space.....	69
Figure 6.6 – DRPEC Name Server Tab.....	70
Figure 6.7 – DRPEC Security Management Tab.....	71
Figure 6.8 – DRPEC User Management Tab.....	72

Acknowledgments

I would like to express my sincere thanks to my supervisor Dr. T. Muldner for his precious guidance, inspiration and valuable time throughout this work. Thanks also extend to Dr. Ke Qiu for being my internal examiner and Dr. Carolyn R. Watters for being my external examiner.

Finally, I would like to express my greatest gratitude to my brother and wife for their love, support, and encouragement.

ABSTRACT

This thesis describes an integrated Distributed Information System (DIS). The first objective of my thesis is to build a portable and distributed information system based on a domain of homogenous and persistent objects. In order to meet this objective, we designed and implemented Distributed Information System (DIS). DIS is a general-purpose environment for the self-sustaining information systems. Software developers can use DIS to create a concrete information system without having to deal with networking and distribution details such as remote access, migration, replication and distributed transactions.

Implementation of DIS is based on a client/server paradigm that uses Java 1.2 and RMI to provide network and operating system independence. Persistent storage on the server is provided through a file system or JDBC.

The second objective of my thesis is to create an experimental system that can be used for the example-based learning. This objective has been met by using DIS to build Distributed Repository of Programming Examples (DRPE).

Chapter 1

1 Introduction

Distributed Information System (DIS) is a general-purpose environment for self-sustaining information systems. DIS enables software developers to create an information system without having to deal with networking and distribution details such as remote access, migration, replication and distributed transactions.

Distributed systems have evolved because the source of the data is centralized and there is often a need for frequent and immediate access to locally generated data. Centralized systems have a potential single point of failure. Distributed systems offer higher overall fault tolerance such that in the event of a failure some or all of the functions of an organization can continue to a greater or lesser degree.

CHAPTER 1. INTRODUCTION

A distributed information system can be defined as a system where documents containing information are distributed across multiple machines connected by a network. Therefore, data (or, documents) are accessible as a shared resource, see [Booth 81]. These systems are useful because the collective storage of multiple computers provides a more powerful system. Additionally, with the duplication of resources the failure of one component does not necessarily imply losing the entire set of data. Thus, distributed systems provide parallelism and fault tolerance, making them potentially much more powerful than their individual components, see [Mullender 89].

In this thesis, we describe the design and implementation of a robust, integrated, persistent distributed information system (DIS). A DIS consists of a number of local information systems (LIS). Each LIS can work locally as a stand-alone application managing its own data, or in conjunction with the name server sharing its data. Each LIS in a DIS provides high-level services to the other LIS, which can be either a client or server. Since each LIS consists of a client and a server, they can join and leave the DIS dynamically. When an LIS comes on-line, it registers its services with a DIS name server, which itself is a special kind of server. The DIS name server is an essential component of DIS that enables information systems to become self-sustaining. The name server is used to dynamically locate other users on the network. It is also used for authenticating other local information systems.

CHAPTER 1. INTRODUCTION

Key responsibilities of the DIS are storage, retrieval and distribution of information and efficient access to the distributed information. The DIS components will communicate with each other via the existing hardware and software network.

A number of applications can be implemented using DIS. Some of them are a virtual office, shared text space, distributed repository of programming examples and virtual class environment.

The second objective of my thesis is to design and implement Distributed Repository of Programming Examples (DRPE). The communication layer is based on RMI and the system is portable, and can be re-implemented using any other distributed technology. The application also provides an efficient and effective way to store, retrieve and manipulate the information.

The GUI is designed and implemented in such a way that it is not only efficient and effective but also easy to learn and use. Dynamic Internet Protocol (IP) addressing is the key concept, enabling a user to connect from virtually anywhere. In my thesis, I show how DRPE can be used to teach programming in C. This part of the thesis has also been described in a separate paper [Muldner, Shiv 00].

There are several existing systems that support example-based learning, see [Neal 89] [Ulf, Raymond 97]. However, our system is innovative because it supports example-based learning in a distributed environment, such as the electronic campus at Acadia University. DIS is

CHAPTER 1. INTRODUCTION

also better than other technologies; for example a File Transfer Protocol (FTP) does not support the same security model as DIS, and does not provide an individual user authentication. There is no way to share some part of the information with a particular client. All the information in an FTP client is accessible either to everybody or to nobody. Web would be useful to implement the DIS, but in this case to download information from the client one would need a digital signature installed on the each and every client.

We assume that the reader has a basic knowledge of Java and RMI [Sun 98, RMI 97]. In my thesis, I used a Courier font for the Java classes and their implementation.

1.1 Overview of the Thesis

The organization of the thesis is as follows. Chapter 2 gives an overview of client/server technology. Chapter 3 gives an overview of distributed technologies, such as CORBA and RMI. Chapter 4 describes the functionality of the DIS. Chapter 5 describes in detail the design and implementation of the DIS. Chapter 6 describes a application of DIS for teaching programming in C: Distributed Repository of Programming Examples (DRPE) and provides several screenshots of DIS. Finally, in Chapter 7, we sum up the conclusions and recommendations for future work.

Chapter 2

2 Client/Server Architecture

The term “client/server” [Adler 95] was first used in the 1980s in reference to personal computers (PCs) on a network with the actual client/server model gaining acceptance in the late 1980s. The client/server software architecture is a versatile, message-based and modular infrastructure intended to improve usability, flexibility, interoperability and scalability as compared to centralized, mainframe, time sharing computing. A client is defined as a requester of services and a server is defined as the provider of services. A single machine can be both a client and a server depending on the software configuration [Schussel 96, Edelstein 94]. When a client needs information from a

server, it requests the information from the server by sending the server a service request. The server processes the request and provides the requested information back to the client.

2.1 Client/server Model

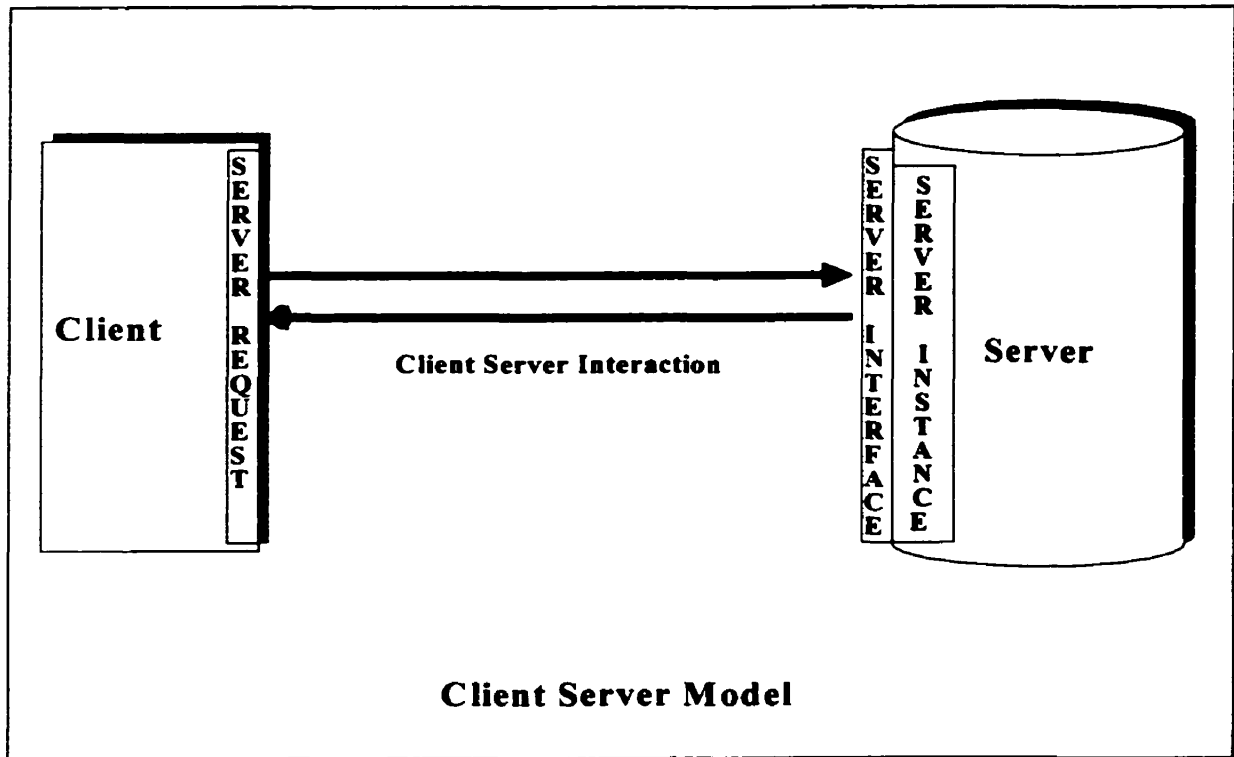


Figure 2.1- Client/server Model.

The client/server model provides a way for different devices to work together, each doing the job for which it is best suited. The role is not fixed, however. A workstation can be a client for one task and a server for another. The client/server model represents various components and interaction procedures (see Figure 2.1) and offers the potential to use resources to their fullest while also facilitating resource sharing. The

CHAPTER 2. CLIENT/SERVER ARCHITECTURE

client/server model fits well in an environment of diverse computing needs that are distributed throughout an organization. In today's business environments, it is expected that client/server architectures will continue to increase in popularity and sophistication. The following subsections describe the components of client/server model.

2.1.1 Client

The client process is usually the front-end of the application that interacts with the users and manages local resources such as the monitor, keyboard, and mouse. Client is also defined as a requester of the services. The client process also contains solution-specific logic and provides the interface between the user and the application system.

2.1.2 Service request

The client makes the service request, a server performs it and the result is returned to the client. Service request sends messages to a server process (program) requesting the server to perform a specific task (service). Service request operates on the user's machine and takes care of the interactive processing driven by the user.

2.1.3 Client/server interaction

Client/Server interaction consists of one or more service requests.

2.1.4 Server

The server performs a function at the request of other application components. A server provides services to other clients that may be connected to it via a network. The connection between client and server is normally by means of message passing, often over a network, and uses some protocol to encode the client's requests and the server's responses. The server may run continuously (as a daemon), waiting for service requests to arrive or it may be invoked by some higher level daemon which controls a number of specific servers.

2.1.5 Service instance

Service Instance is a combination of software and data that provides services and maintains the context and state specific to it. A service instance may be statically defined or dynamically created and destroyed at run-time. Its life-time may be long (years) or short (sub-seconds).

2.1.6 Service interface

This is an abstraction that represents externally visible behavior of a service instance.

2.1.7 Service reference

A service reference points to a service instance.

2.2 Client/server Communication

To ensure proper interaction between clients and servers a new type of software called “middleware” [Benda 97] has been developed. Middleware is also referred to as communication layer. One of its purposes is to translate client requests into a form that servers can understand and then translate server responses for clients. Middleware is the key to delivering resilient, secure and transparent services to users. It is a layer of software that runs between the client and the server processes. It shields the client from the complexity of underlying communications protocols, network operating systems and hardware configurations. Several types of middleware services are available such as RMI, RPC, RDA, CORBA and DCOM.

RPC, RMI, DCOM, CORBA or some other variant is widely used for client/server communication in a distributed systems environment. The format of communication between clients and servers takes in the form of message exchanges. The simplest exchange consists of a request message from a client to a server and a reply message from the server to the client. Each communication takes the form of a single message transmitted between processes.

2.3 Client/Server responsibilities

Client/server responsibilities can be defined into three different groups. These groups identify the responsibilities of client, server and middleware as follows:

2.3.1 Client Responsibilities:

- Provide user Interface.
- Translate the user's request into the desired protocol.
- Transmit the request to the server.
- Wait for the server's response.
- Translate the response from the server back to the client.

2.3.2 Server Responsibilities:

- Listen for a client's request.
- Process that request.
- Return the results to the client.

2.3.3 Middleware Responsibilities:

- Middleware forwards the client's request such that the server can understand and translate server responses for the clients.

2.4 Tiers

In general client/server architectures now have three tiers. The first-tier, or top-tier, includes a client with user system interface where user services (such as session, text input, dialog, and display management) reside [Louis 95]. The middle-tier, or middleware, provides process management services (such as process development, process enactment and process monitoring) that are shared by multiple applications. The third-tier provides database management functionality and is dedicated to data and file services that can be optimized using any proprietary database. The data management component ensures the data is consistent throughout the distributed environment through the use of features such as data locking, consistency, and replication. It should be noted that connectivity between tiers could be dynamically changed depending upon the user's request for data and services.

In the two-tier client/server model, the middle-tier services are usually moved onto the client side. This is a typical two-tier client/server architecture, fat client and thin server. For three-tier client/server architecture, we move the functionality part from the client to another platform, leaving it as a thin client and a thin server.

2.4.1 Two-tier Architecture

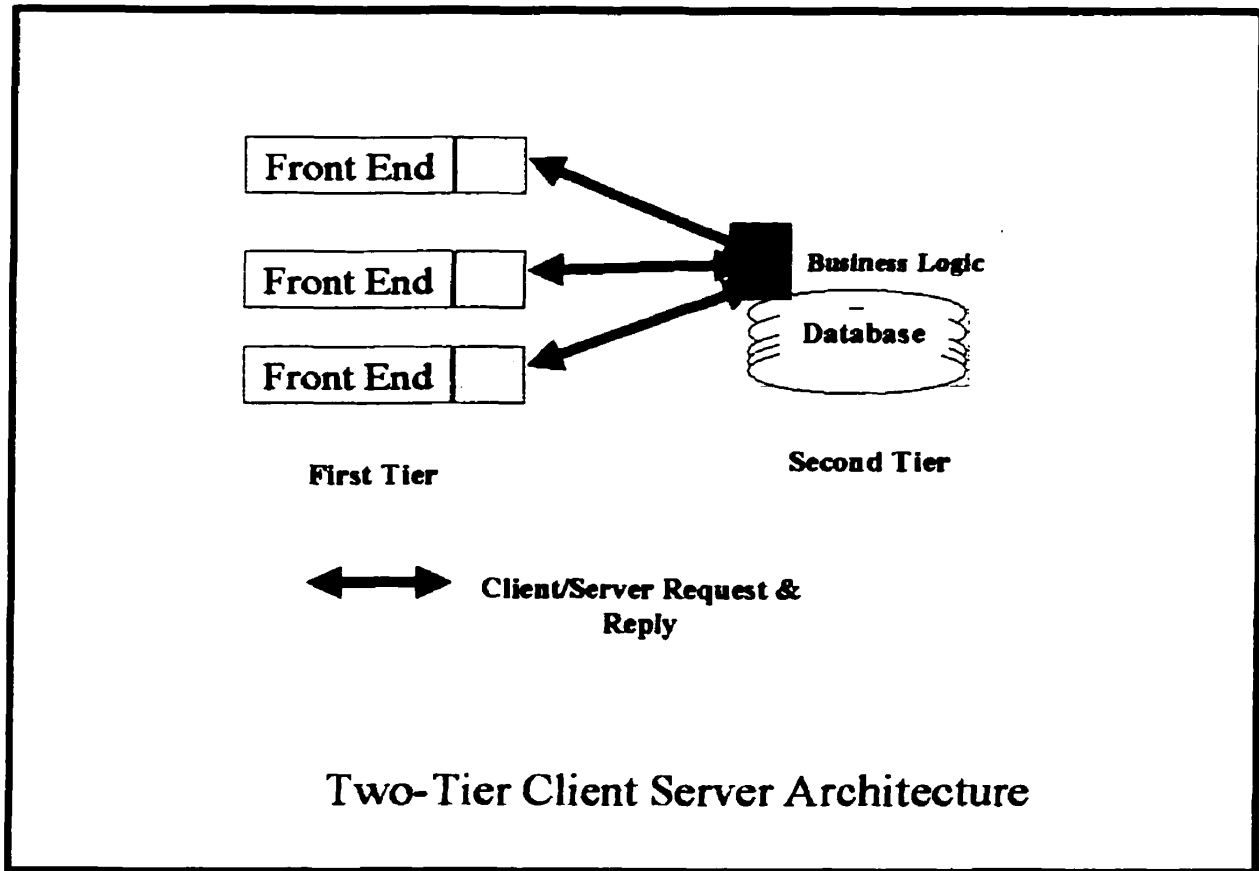


Figure 2.2- Two-tier Architecture.

With two-tier client/server architecture (see Figure 2.2), the first-tier, or system interface, is usually located in the user's desktop environment and the second-tier, or database management services, are usually at a server. The processing management, is split between the user system interface and the database management tier.

The two-tier client/server architecture is a good solution for distributed computing when work groups are defined as a dozen to 100

CHAPTER 2. CLIENT/SERVER ARCHITECTURE

people interacting on a LAN simultaneously. It has, however, a number of limitations. When the number of users exceeds 100, performance begins to deteriorate. This limitation is a result of the server maintaining a connection via "keep-alive" messages with each client, even when no work is being done. A second limitation of the two-tier architecture is that implementation of processing management services by using vendor proprietary database restricts flexibility and choice of DBMS for applications. Finally, current implementations of the two-tier architecture provide limited flexibility in moving (repartitioning) program functionality from one server to another without manually regenerating procedural code [Schussel 96, Edelstein 94].

2.4.2 Three-tier architecture.

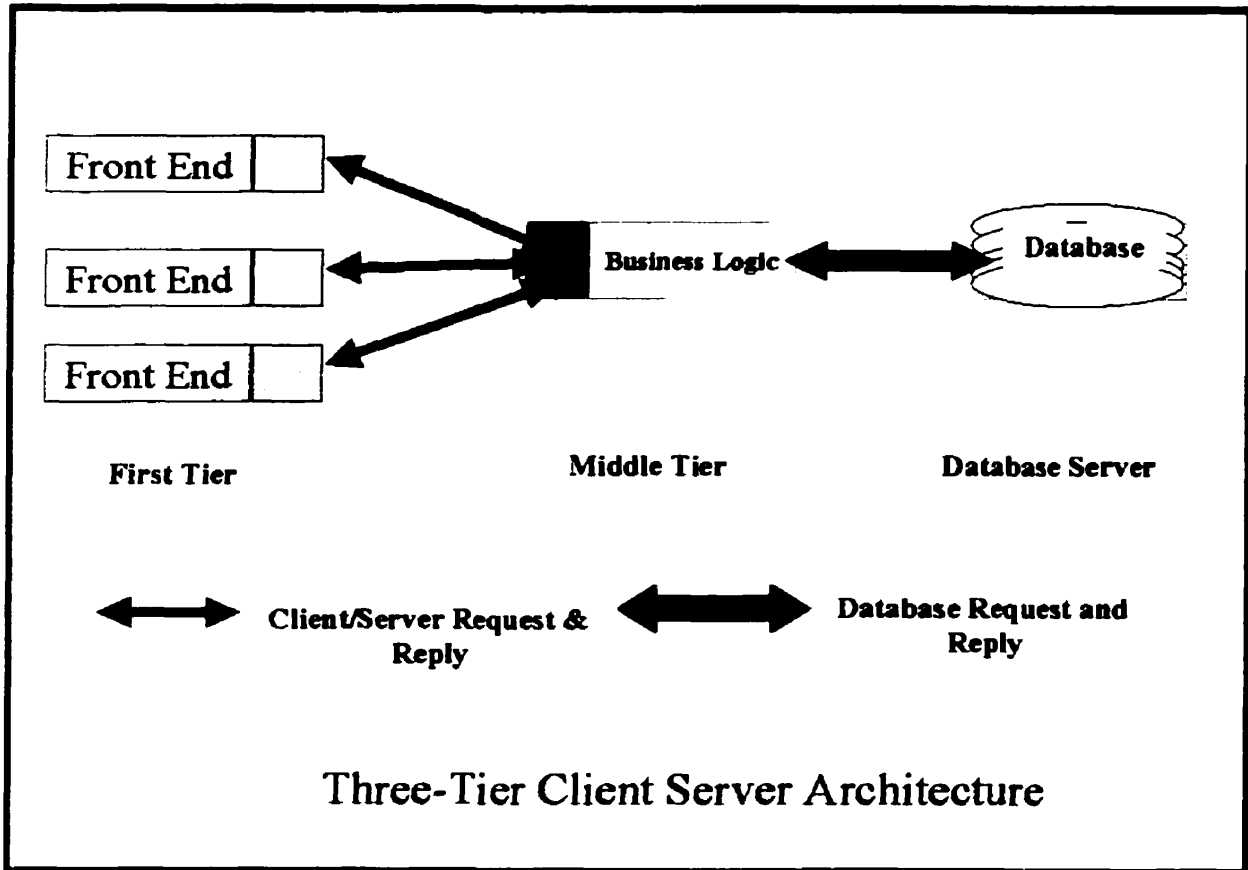


Figure 2.3- Three-tier architecture.

The three-tier architecture (see Figure 2.3) emerged to overcome the limitations of the two-tier architecture. In the three-tier architecture, a middle-tier was added between the user system interface and the database management server. There are a variety of ways of implementing this middle-tier, such as transaction processing monitors, message servers, or application servers. The middle-tier performs various services like queuing, application execution, and database staging. For example, if the middle-tier provides a queuing service, the client can

CHAPTER 2. CLIENT/SERVER ARCHITECTURE

deliver its request to the middle layer and disengage because the middle-tier will access the data and return the answer to the client. The three-tier client/server architecture has been shown to improve performance for groups with large numbers of users (in the thousands) and improves flexibility when compared to the two-tier approach. Flexibility in partitioning can be as simple as "dragging and dropping" application code modules onto different computers in some three-tier architectures. A limitation with three-tier architecture is that the development environment is reportedly more difficult to use than the development of two-tier applications [Schussel 96, Edelstein 94].

2.5 Expected Benefits of Client/Server Computing

The client/server model provides many benefits. An explanation of those benefits follows:

Adaptability: - Client/server computing has the ability to adapt to the changing needs of the business environment, to ease up or downsize the computing resources to match the business needs.

Reduced Operating Costs: - Client/server computing reduces hardware and software costs which means real computing power is increased. Client/server computing means large expensive systems can be replaced by lower cost smaller ones, networked together.

Platform Independence: - The trend to client/server computing goes hand in hand with the push towards open systems and industry standards. No one wants to be locked into a single vendor's propriety hardware or software. Users want to be able to freely interchange components.

Better Return on Computing Investment: - A client/server environment provides vendor independence and allows computing resources to be purchased freely.

Improved Performance: - Client/server processing power spreads through-out the organization giving users faster response times. Using

CHAPTER 2. CLIENT/SERVER ARCHITECTURE

open networked systems and lower component costs, new resources can be added quickly where needed to improve performance bottlenecks.

Decentralized Operations: - A client/server decentralized IT operation puts computing power and data access in the hands of the users. This increases the productivity of MIS staff by reducing trivial requests. Client/server architectures can improve the service provided to customers by supplying information at the point where it is required for customer requests.

High Reliability: - Client/server operations require highly reliable systems, with high transaction rates, timely and continuous data access, data integrity and corporate security.

2.6 Summary

A client is defined as a requester of services and a server is defined as the provider of services. A single machine can be both a client and a server depending on the software configuration.

The Client/Server architecture model is a versatile, message-based and modular infrastructure intended to improve usability, flexibility, interoperability and scalability as compared to centralized, mainframe, time sharing computing. The Two-tier client/server architecture is used extensively in non-time critical information processing where management and operation of the system are not complex. The two-tier

CHAPTER 2. CLIENT/SERVER ARCHITECTURE

architecture works well in relatively homogeneous environments where processing rules (business rules) do not change often and workgroup size is expected to be fewer than 100 users. The three-tier architecture improves performance, flexibility, maintainability, reusability and scalability by centralizing the process logic. The centralized process logic makes administration and change management easier by localizing system functionality so that changes must only be written once to be available throughout the system. With other architectural designs, a change to a function (service) would need to be written into every application [Eckerson 95].

Chapter 3

3 Distributed Computing

Today's software development projects are targeted for heterogeneous computing environments that integrate new systems with legacy components. The distributed computing architecture enables application developers to benefit from the use of this technology.

As networks of computing resources have become prevalent, the concept of distributing computing over multiple resources has become increasingly viable and desirable. Over the years, several methods have evolved to enable this distribution, ranging from simplistic data sharing to advanced systems supporting a multitude of services. This chapter

presents an overview of distributing computing, covering core technologies and their benefits.

3.1 Advantages of Distributed Computing

Distributed computing supports development in heterogeneous environments. Today's software applications have complex requirements often requiring the use of many types of computers and tools such as GUI builders, desktop computers, servers, etc. Distributed computing provides a foundation for using these tools and systems together. It provides freedom to select from a wide range of hardware, software and networking components. Here are some of the advantages of distributed computing:

- A greater cost-effectiveness through sharing of computing resources and implementations of heterogeneous open systems.
- It provides collaboration through connectivity and internetworking.
- Better performance through parallel processing.
- Scalability and portability through modularity. Distributed computing allow corporations to deliver fully scalable, completely networked applications that can be delivered on any type of network, including LANs, WANs and the Internet. This capability allows an application to be utilized in a variety of ways. One of the most popular methods of using distributed applications is via a browser-based interface.

because it allows access at any time from virtually anywhere in the world.

- Increased reliability and availability through replication.
- Extensibility through dynamic configuration and reconfiguration.

3.2 Technologies Overview

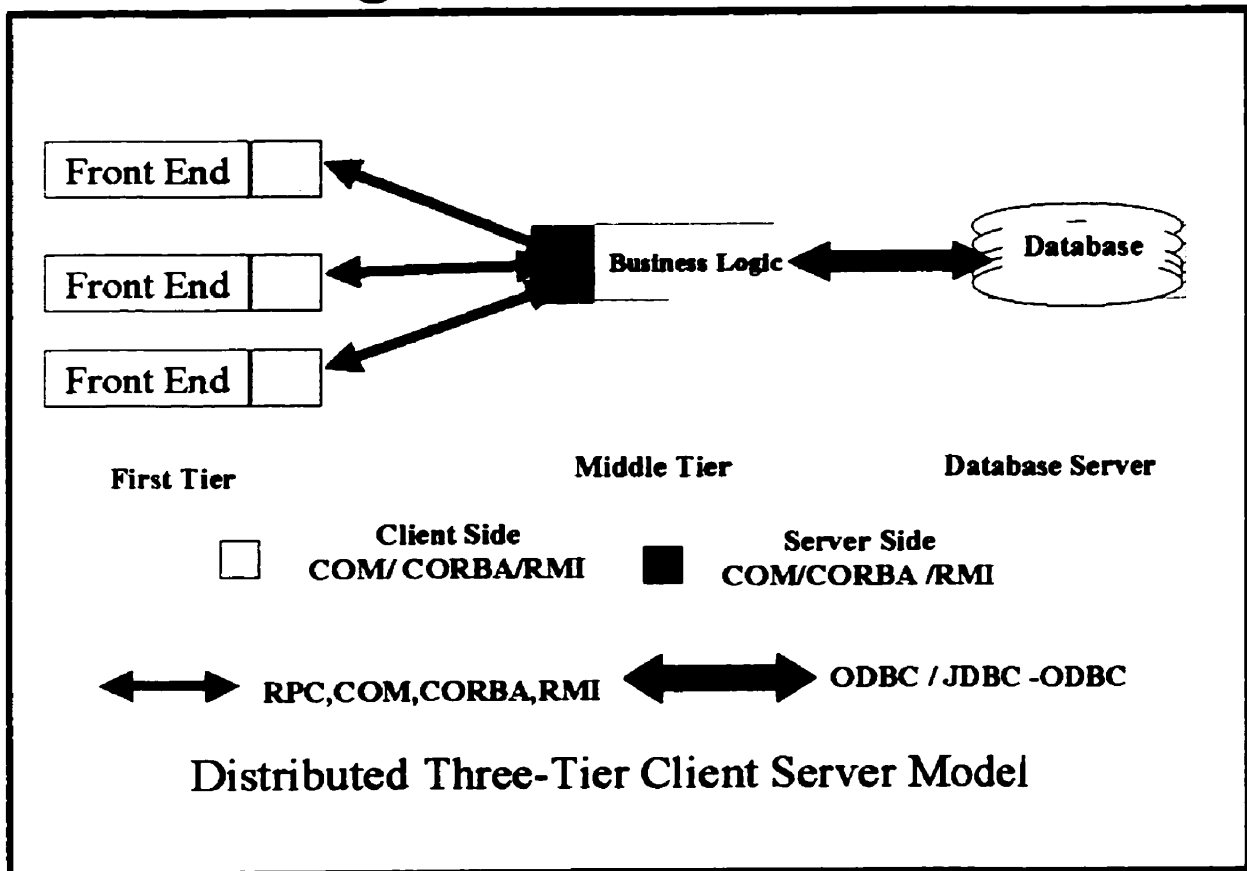


Figure 3.1 – Distributed Architecture.

The research for heterogeneous computing environments led to the development of distributed computing standards such as the Distributed Computing Environment (DCE). The DCE specification is among the

CHAPTER 3. DISTRIBUTED COMPUTING

most widely implemented in the industry, providing consistent behavior across heterogeneous execution environments. The DCE architecture also defines thread, time, authentication & security and directory & naming services. These standards are followed by the Distributed Component Object Model [DCOM 97] from Open Software Foundation and Open Group [Microsoft 97], the Common Object Request Broker Architecture [CORBA 97] from the Object Management Group [OMG 98] and Remote Method Invocation (RMI 97) from Sun Microsystems [Sun 98]. Each has its own advantages and disadvantages. In the rest of this chapter, a brief overview is given for each of these technologies with regards to choosing a particular technology for building DIS.

3.2.1 DCOM - Distributed Component Object Model

Microsoft's core object distribution protocol is DCOM [DCOM 97], an extension of Microsoft's Component Object Model [COM 95] integration architecture, permitting interaction between objects executing on separate hosts in a network.

In order to address the rising need for distribution of objects across multiple hosts (i.e. multiple physical address spaces), Microsoft developed DCOM as an extension to COM. As an extension rather than a separate architecture, DCOM inserts a stub interface between the calling application and the actual implementation of that interface. In this manner the architecture strongly resembles an RPC-based model, although the implementation is still based on a binary integration scheme, rather than a more abstract model.

The DCOM does not support distributed naming services, rather it is based on the NT registry. Configuring and installing DCOM is tedious and labor intensive job. Although, DCOM is well suited on a Microsoft platform, it is not for the other vendor's platforms.

3.2.2 CORBA - Common Object Request Broker Architecture

CORBA is a standard maintained by the Object Management Group [OMG 98] for the distribution of objects across heterogeneous networks. Designed as a platform-neutral infrastructure for inter-object communication, it has gained widespread acceptance. CORBA allows applications to use a common interface, defined in an Interface Definition Language (IDL), across multiple platforms and development tools. OMG IDL is designed to be platform and language-neutral; data and call format conversions are handled transparently by the Common Request Broker (ORB). All interfaces to CORBA objects, and the data types used in those interfaces, are specified in the IDL. This common definition allows applications to operate on objects without concern for the manner in which the object is implemented.

CORBA also provides some capabilities for runtime object interface identification and invocation through its Interface Repository (IR) and Dynamic Invocation Interface (DII). While these have the potential to allow (almost) complete runtime configuration to access CORBA objects, in practice there may be very few cases where such capabilities are actually workable due to semantic issues. The implementation of CORBA is a tedious job and requires lot of development time.

3.2.3 Java RMI - Remote Method Invocation

Java Remote Method Invocation [RMI 99a] is a distributed object model for the Java platform. RMI is unique in that it is a language-centric model that takes advantage of a common network type system. In a nutshell, RMI [RMI 99b] extends the Java object model beyond a single virtual machine (VM) address space. Object methods can be invoked between different VMs across a network and actual objects can be passed as arguments and return values during method invocation. Java RMI uses object serialization to convert object graphs to byte-streams for transport. Any Java object type can be passed during invocation, including primitive types, core classes, user-defined classes, and JavaBeans™. The ability to pass actual objects enables clean system design, allowing system designers to focus on the overall object model, not the plumbing of an application. Java RMI could be described as a natural progression of procedural RPC, adapted to an object-oriented paradigm. Java RMI can dynamically resolve method invocations across VM boundaries and it also provides a fully object-oriented (OO) distributed environment. Developers can implement classic OO design patterns for distributed programming just as they would in local programming. Because RMI operates naturally in the Java domain, developers work within a single object model (the Java model) instead of working with multiple object models (Java, CORBA IDL, and others). This

removes a great deal of complexity. Unlike language-neutral object models, RMI requires no mapping to common interface definition languages. The syntax of remote method invocations is almost exactly the same as local method invocations. RMI removes the burden of memory management from the programmer because the underlying system provides distributed garbage collection. All of these characteristics make programming in simple and natural, an obvious choice for developing 100% pure Java client/server, peer-to-peer, or agent-based applications.

RMI also exhibits some distinctive new capabilities. Executable code can be dynamically distributed on demand, including all necessary code for distributed applications (client objects, remote interfaces, and remote object stubs). This means that no code needs to be preinstalled on client machines, greatly reducing the burdens of software distribution and system maintenance. Because the common type system is the Java VM and language environment, RMI works reliably across different operating systems where a Java-compatible VM is available. The RMI system also takes advantage of the secure nature of the Java environment. The combination of Java technology's automatic bytecode verification, secure loading of classes at runtime and disallowing access to memory pointers, makes Java RMI secure from the ground up.

3.3 Advantages of RMI

This section describes some of the advantages associated with RMI:

Object Oriented: RMI can pass full objects as arguments and return values, not just predefined data types. This means that we can pass complex types, such as a standard Java hashtable object, as a single argument. In existing RPC systems the client would have to decompose such an object into primitive data types, ship those data types and then recreate a hashtable on the server. RMI lets you ship objects directly across the wire with no extra client code.

Mobile Behavior: RMI can move behavior (class implementations) from client to server and server to client. For example, we can define an interface for examining employee benefit reports to see whether they conform to current company policy. When a benefit report is created, the client can fetch an object implementing the report from the server. When the benefits policies change, the server will start returning a different implementation of the interface using the new policies. The constraints will therefore be checked on the client side, providing faster feedback to the user and less load on the server, without installing any new software on the user's system. This provides maximum flexibility, since changing policies requires us to write only one new Java class and install it once on the server host.

Safe and Secure: RMI uses built-in Java security mechanisms that allow your system to be safe when users download the implementations. RMI uses the security manager designed to protect systems and networks from hostile applets and from potentially hostile downloaded code. In severe cases, a server can refuse to download any implementations at all.

Easy to Write/Easy to Use: RMI makes it simple to write remote Java servers and Java clients that access those servers. A remote interface is an actual Java interface. A server has roughly three lines of code to declare itself a server and otherwise is like any other Java object. The simplicity allows for quick and easy writing of servers for full-scale distributed systems quickly. It also permits rapid prototyping and designing early versions of software for testing and evaluation. Finally, because RMI programs are easy to write they are also easy to maintain.

Connects to Existing/Legacy Systems: RMI interacts with existing systems through Java's native method interface JNI. Using RMI and JNI we can write the client in Java and use the existing server implementation. When we use RMI/JNI to connect to existing servers we can rewrite any parts of the server in Java. Similarly, RMI interacts with existing relational databases using JDBC without modifying any existing non-Java source that uses the databases.

Write Once, Run Anywhere: RMI is part of Java's "Write Once, Run Anywhere" approach. Any RMI based system is 100% portable to any

Java Virtual Machine. If you use RMI/JNI or RMI/JDBC to interact with an existing system, the code written using JNI or JDBC will compile and run with any Java virtual machine.

Distributed Garbage Collection: RMI uses its distributed garbage collection feature to collect remote server objects that are no longer referenced by any clients in the network. Analogous to garbage collection inside a Java Virtual Machine, distributed garbage collection lets you define server objects as needed, knowing that they will be removed when they are no longer needed.

Parallel Computing: RMI is multi-threaded, allowing servers to exploit Java threads for better concurrent processing of client requests.

3.4 Summary

Certainly, of the three frameworks discussed, CORBA provides the greatest flexibility with its language and platform neutrality. There are, of course, some costs associated with this neutrality, both in deployment and runtime overhead.

Microsoft's COM/DCOM solution is the Windows operating system installed base providing a compelling argument for its use in Windows only environments.

Java RMI provides a language-specific architecture allowing Java-to-Java distributed applications to be built easily. The main advantage to

CHAPTER 3. DISTRIBUTED COMPUTING

using Java RMI when designing a pure Java distributed system is that the Java object model can be taken advantage of whenever possible. Of course, this precludes using Java RMI in multilingual environments. Java's inherent platform independence, however, still allows deployment in heterogeneous environments.

Many of the above concepts are shared across distribution architectures. Some technologies, yet old, still offer compelling reasons to use them. However, where ease and cost of deployment are larger factors, RMI is generally a good choice.

The three-tier architecture has improved performance and flexibility as compared to the two-tier approach and Java RMI is the best-suited distributed environment for use in the distributed information system.

Chapter 4

4 Introduction of DIS

The main objective of this project is to build a portable, distributed, persistent information system. A distributed information system can be defined as a system where documents containing information are distributed across multiple machines connected by a network. Therefore, data (or, documents) are accessible as a shared resource.

Our most basic objective is to exchange and share information. From now on, we will assume that the information is stored in a document, which serves as a persistent medium for this information (we will use the the term “document” interchangeably with the term “information”). A

document is not necessarily the same thing as a file; it could be a file, a number of files logically grouped together, or an entry in the database.

4.1 Push and pull model

DIS is based on the pull model. In order to describe the task of exchanging and sharing information in a pull model, we consider two kinds of applications, here called providers and fetchers respectively. A provider application gives access to the available information for authorized users; a fetcher application is designed to fetch or browse information from one or more providers. Typically, a provider is implemented as a server, and a fetcher as a client. For more information about clients and servers see [Mullender 89]. A single application may be both a provider (server) and a fetcher (client) at the same time. In DIS each client is a provider as well as a fetcher. Further extension can provide the pull model.

4.2 Main Functionality

DIS is a collection of documents (files), in which every user can define her or his classifications (folders). Each document may be stored or classified within one or more classifications, and classifications may be nested. Therefore, in a classical file system, classifications resemble folders, or directories, and documents resemble files. The entire system

CHAPTER 4. DESCRIPTION & FUNCTIONALITY OF DIS

can be seen as a tree; leaves, however, may have more than one parent (classification). The user will be able to make a part of his or her system available to other users and pull (download) a classification from another user (this classification corresponds to the node of the tree and the entire subtree rooted at this node may be pulled). Permissions may be set so that only selected users will have access to some documents.

DIS is an application designed to satisfy the above requirements. In this system, a single unit that resembles a file represents each document and a folder represents a classification. DIS documents are stored separately from the file system, but they can be easily imported from and exported to that system.

DIS is a program that can operate both in off-line mode or in on-line mode, and in the latter mode as a server or as a client. All of the operations that can be performed in off-line mode can also be performed in on-line mode.

4.3 Component of DIS

DIS is a collection of various local information systems (LIS) and a name server. DIS works in conjunction with the services provided by the various Local Information Systems (LIS) and a name server. The different components of the distributed information system (DIS) are as follows:

- DIS Name Server
- Local Information System (LIS)

4.3.1 Local Information System (LIS)

Local Information system is an individual entity that can work locally by itself (Off-line Mode) or in conjunction with other LIS and a name server (On-line mode). LIS works both as client and server when it works in on-line mode. LIS can leave and join the DIS environment dynamically.

Components of the local information system (LIS) are as follows:

- File
- Folder

4.3.1.1 File

A File represents a document in a distributed information system. A File is the smallest atomic structure stored in the local information system. A DIS file is stored separately from the file system.

Various operations that can be performed on the file are as follows:

- Adding a new file.
- Deleting an existing file.
- Modifying an existing file.
- Copying the file to a folder.
- Exporting a file to DIS.
- Importing a file from DIS.
- Moving the file.

4.3.1.2 Folders

A Folder represents the classification in a distributed information system. A Folder is the collection of sub folders and files. Folders are used to define the classification within the system. Initially at the top level there exist three placeholders (downloadspace folder, connected workspace folder and workspace folder). These are explained below:

4.3.1.2.1 Workspace Folder

This is one of the main folders that contains other subfolders. This folder participates in the on-line mode of operations of the DIS. This folder provides the classifications that are accessible from the other users based on user privileges. Each subfolder can contain additional files or sub folders. Each folder in a workspace folder has an attribute attached to it, a permission attribute. Permissions are the lists of users

that have rights to see all files in a current folder but not necessarily in any subfolders. The owner of the system sets these permission attributes.

4.3.1.2.2 Downloadspace folder

This is a temporary workspace folder. All the classifications within this folder are not accessible to any other users.

4.3.1.2.3 Connected workspace folder

The connected workspace folder is used when the user wants to fetch documents or classifications from the network on his or her local system. The user gets connected to the other user(s) on the network and loads the information into his or her connected workspace area.

The following operations can be performed on the folders:

- Add a new folder.
- Delete a folder.
- Move a folder and all its sub components to another location.
- Modify the permission attributes of a folder.

4.3.2 DIS Name Server

The DIS name server is the central repository for storing user ID and password for every user. It is also used to store the dynamic IP address of the user. For an application "A" to communicate with another application "B", "A" must be able to locate "B" using some kind of naming system. The standard convention used by the Internet is the Universal Resource Locator (URL). When "A" provides "B's" URL, the Domain Name Server, DNS, finds "B's" IP address and now "A" can use this address to communicate with "B". Unfortunately, this technique does not work if "B" is off-line or if "B" uses an Internet Service Provider (ISP) which provides dynamic IP addresses. Therefore, often we assume that there exists an additional name server residing at a "well-known" static IP. When the application "B" goes on-line, it connects to the name server which then retrieves "B"'s current IP and associates it with that application's name. When application "A" wants to connect to application "B", it does so through the name server, which guarantees that "B" is on-line and its current IP address is known.

The main functionalities of the name server are user validation, dynamic IP addressing and handling client requests. Since a user can virtually login from any terminal, he or she has to update their current IP address and download all the active users. After downloading the active

users, the user can login to any active user after connecting to its IP address obtained from the name server.

An administrator of this name server manages users, (i.e. registers the users, assigns passwords, etc). The name server is running on a computer with a “well-known” static IP address, which is provided in the configuration file read at the application startup. The administrator manages the name server including adding and removing users, and setting their passwords.

The current implementation of the name server has one major drawback as we have only one name server running, and if this name sever is down the user would not be able to connect to other LIS's. We will address this issue in our future work.

4.4 Operations

4.4.1 Off-line operations

Off-line operations facilitate organizing a repository (i.e. creating new classifications, modifying and deleting existing classifications, importing and exporting, and browsing and viewing documents). The user achieves this through the use of menus, right button, or drag and drop for these operations.

CHAPTER 4. DESCRIPTION & FUNCTIONALITY OF DIS

Also in off-line mode, the user may specify "permissions" (i.e. give or revoke the right to access documents to other users). Here, a classification is the smallest unit the user can grant permission to then these users can view and possibly pull all documents in this classification. However, their ability to view the contents of any nested classifications depends on their permissions for these nested classifications.

4.4.2 On-line operations

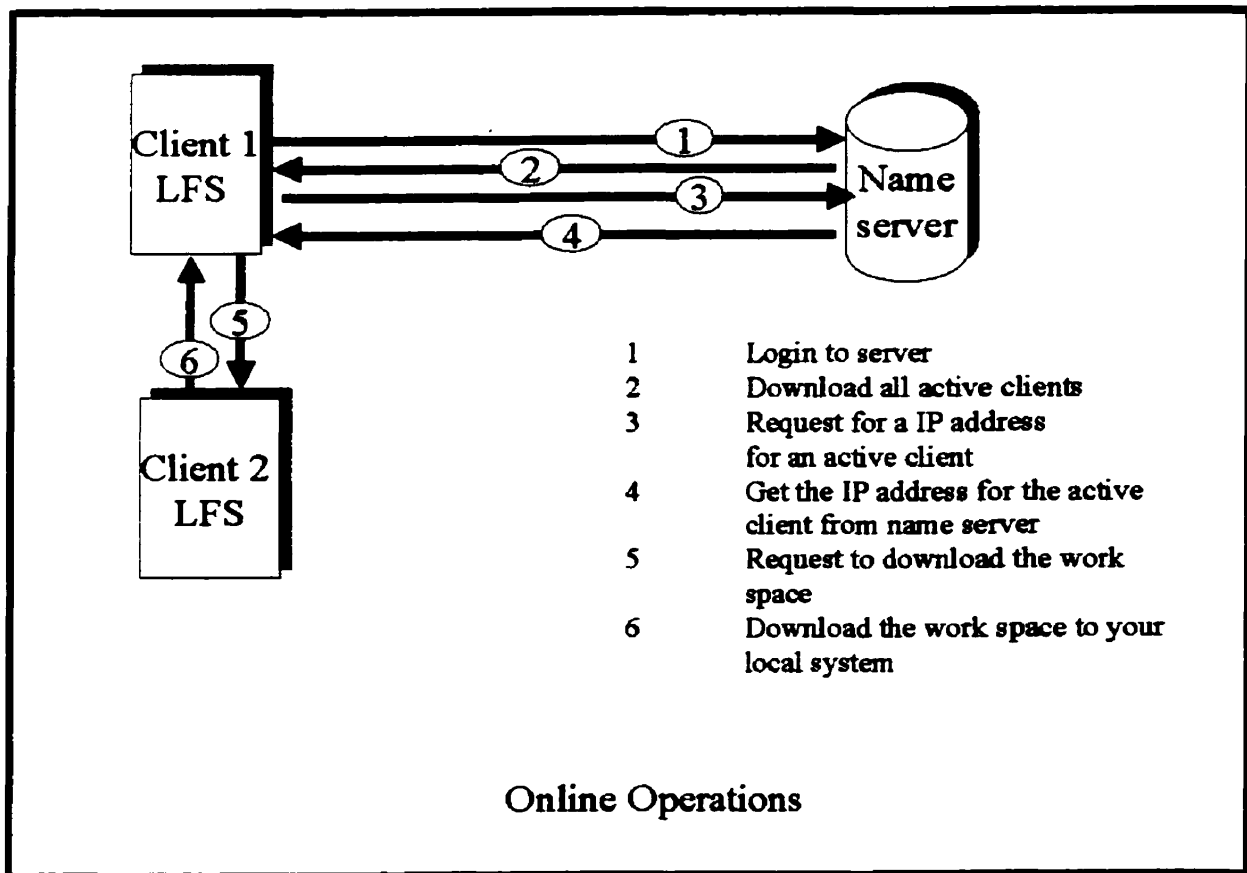


Figure 4.1 - Sample Online operations.

On-line operations can be divided into two types: interactions with the name server and interactions with other users. In order to interact with the name server the user must login to the name server by providing valid name and password. Upon successful login the name server retrieves the current IP of this user and saves this information. DIS, therefore, supports users who have dynamic IP addresses such as those using ISPs. The user may perform the following operations that interact with the name server: change the password, retrieve the list of all

CHAPTER 4. DESCRIPTION & FUNCTIONALITY OF DIS

registered users, or retrieve the list of all active users (i.e., users who are currently on-line). The list of active users is used to start an interaction with another user. Connecting to another user is accomplished simply by selecting the user from the active user list. Note that this connection does not require an authentication because both users must have current connection to the name server and, therefore, have both already been authenticated. Also, the name server provides the current dynamic IP addresses of all users. The list of currently active users can be used to select one user and then connect to her or him.

User "A", upon connecting to another user "B", may browse and download the classifications permitted by the user "B" specifically for user "A".

4.5 Security Model

4.5.1 Name server Security model

The name server security model describes the security model on the name server. There are two basic types of accounts on the name server:

- Administrator
- Other users

The administrator can add, modify and delete any user accounts. A user is only permitted to change the password and IP address of his or her account.

4.5.2 LIS Security model

The security model chosen for DIS is a mixture of file level security and the folder level security for the DIS. This means the user is allowed to view and download all the files within folder but not the files in the sub folders or in the parent folder. To achieve this we have a permission attribute attached to every folder which is a list of users. This list of users in the permission attribute is a subset of authentic users on the name server. The users in the permission attribute of folders are authentic to view and download all the files within the folders. The user of LIS can set the permission attribute of his or her own folders. If another user is in the list of the permission attribute of the folder then he or she can view and download the contents of that folder.

4.6 Object Persistence

Object instance needs to persist over time beyond the life cycle of an application. DIS and name server support persistent instance by converting a sequence of bytes stored in a form of long term media. The restoration process creates an environment identical to the original one.

The current persistent form in LIS and the name server is a file in a local file system, but we have a generic interface to provide the flexibility to upgrade to another implementation in the future. This can be a relational database or object oriented databases.

4.7 Exception Handling

Different kinds of error checking and exception handling are being incorporated in DIS. The low-level functions deduce the errors, which are passed to the high-level user interface for display. Some of these exceptions and errors are:

- Trying to perform on-line operations (such as downloading active users, connecting to another user or changing the password) while you are off-line.
- Inactive name server instance.
- User authentication such as invalid user id and password.

Chapter 5

5 Implementation of DIS

In this chapter we will discuss the design and implementation of DIS.

We will go into different phases of the implementation process by using class diagrams and code fragmentation.

5.1 Analysis

5.1.1 Strategy

We define our strategy for building DIS based on the domain of distributed persistent object model of Java 1.2. The system is independent of any particular technology such as computer hardware, operating system, databases or distributed technology. The graphical

user interface is designed and implemented in such a way that it is not only efficient and effective but also easy to learn and use. We try to instill all the application functionality in one main frame with different tabs rather than having separate applications with many screens. This makes the user understand the system easily without extra effort.

5.1.2 Analysis

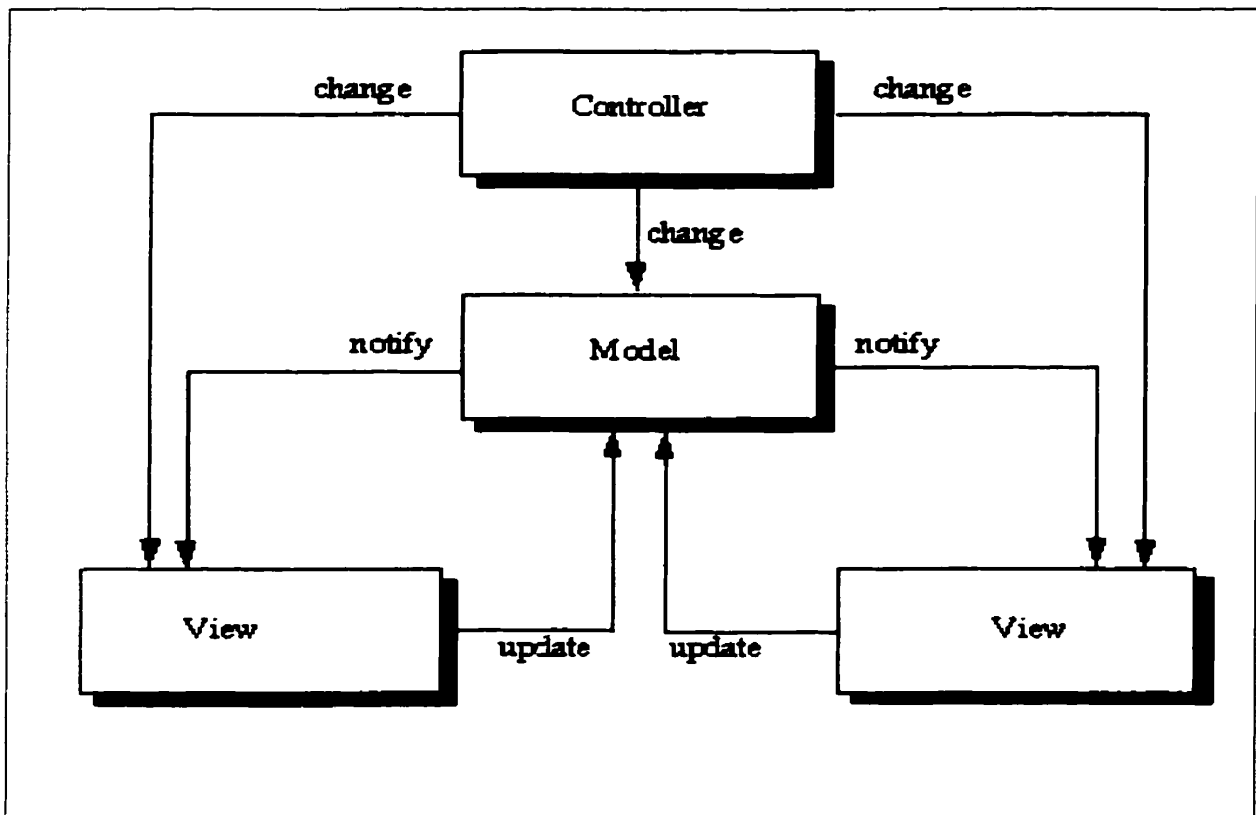


Figure 5.1 – Model View Controller.

The standard design pattern, Model View Controller (MVC), is used in building the graphical user interface. Putting this in the paradigm of DIS, the data (Data Model) in the application is kept separate from the

CHAPTER 5. IMPLEMENTATION OF DIS

rendering of the data (View Model) and the manipulation of the data (Controller Model), (see Figure 5.1). The goal is to separate the application object (model) from the way it is represented to the user (view) and from the way in which the user controls it (controller).

5.2 Design

5.2.1 Architecture

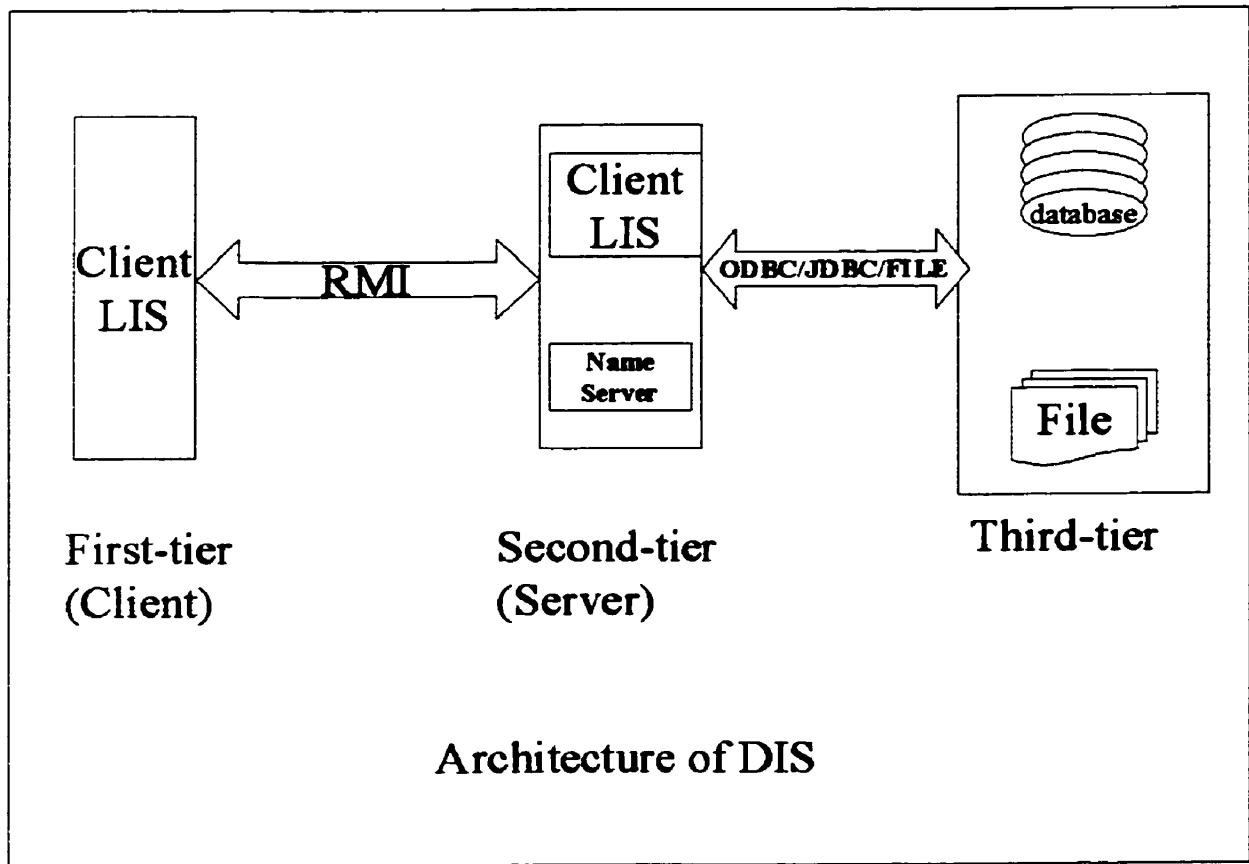


Figure 5.2 - Architecture of DIS.

DIS is a three-tier client server architecture as shown in Figure 5.2. The first-tier represents the Local Information System (LIS). LIS queries the Name server (second-tier) to find the IP of the other registered LIS by sending the service request to the name server. When the name server processes the request, the output values are returned to the requestor LIS. After retrieving the IP, the LIS client object locates the appropriate implementation of the remote object and transmits

parameters and control to the object implementations. Service objects on the server side of the name server or the client may use JDBC/ODBC to communicate with the third-tier, the database server.

5.2.2 Model

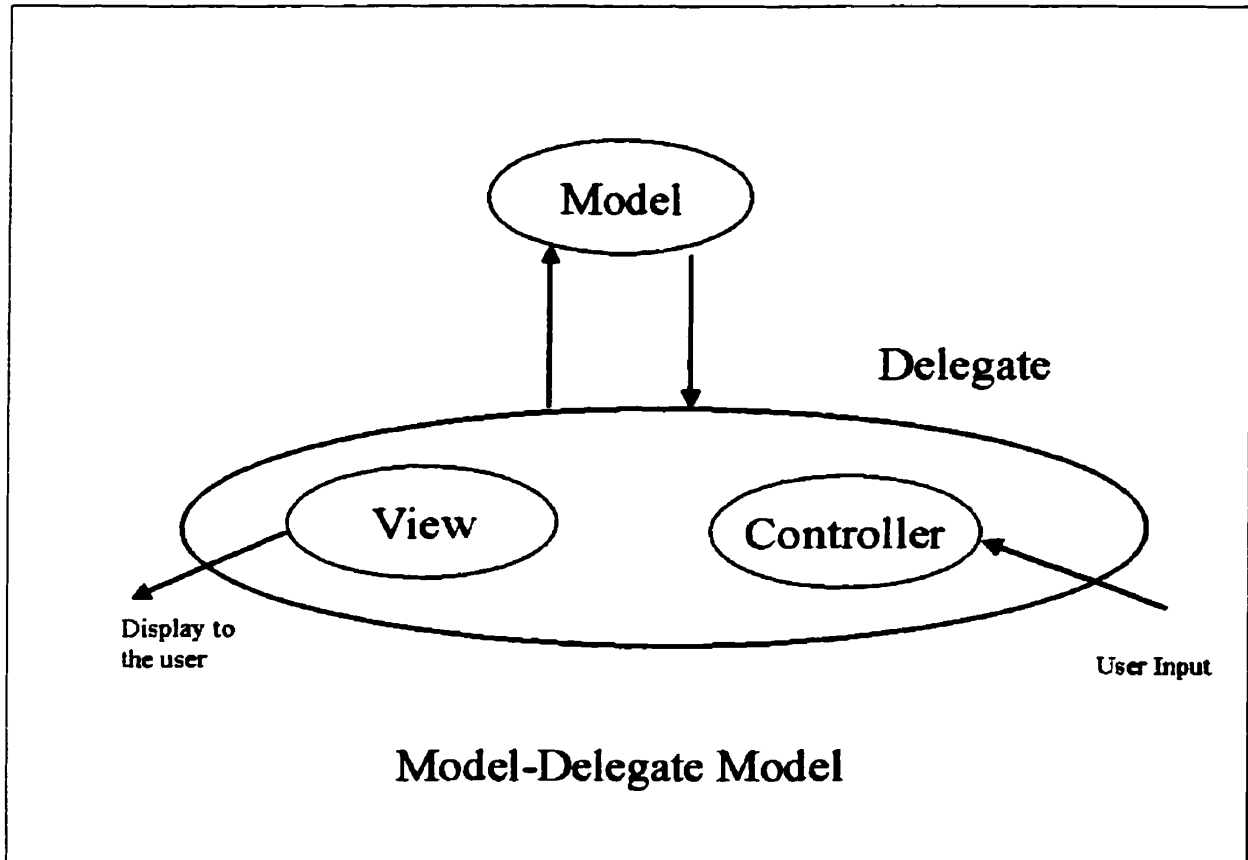


Figure 5.3 – Model-Delegate Design Model.

DIS uses a common variant of MVC where the view and the controller are merged into one piece, the delegate. In practice, the view and the controller are too closely related to be treated separately so merging the view and the controller greatly simplifies the development.

CHAPTER 5. IMPLEMENTATION OF DIS

This variant is called the Document-View (see Figure 5.3). It is also known as the Model-Delegate design.

In this Model-Delegate design, the delegate updates the model through the known interface and the model informs the delegate when to update information through an event listener. The delegate then retrieves the updated information through the same known interface.

Separating the data from the display of the data leads to a great deal of flexibility in the development of the user interface. A major advantage of this model is that a single delegate class can display and update the data from several different models with the only restriction being that the models support a common interface. Another benefit is reduced memory management as only one copy of the data is maintained.

5.3 Implementation

In the discussion of implementation of DIS it is important to distinguish between distributed object programming which allows the methods of the object to be invoked across different address space and simple object oriented programming where the objects are assumed to share the same address space.

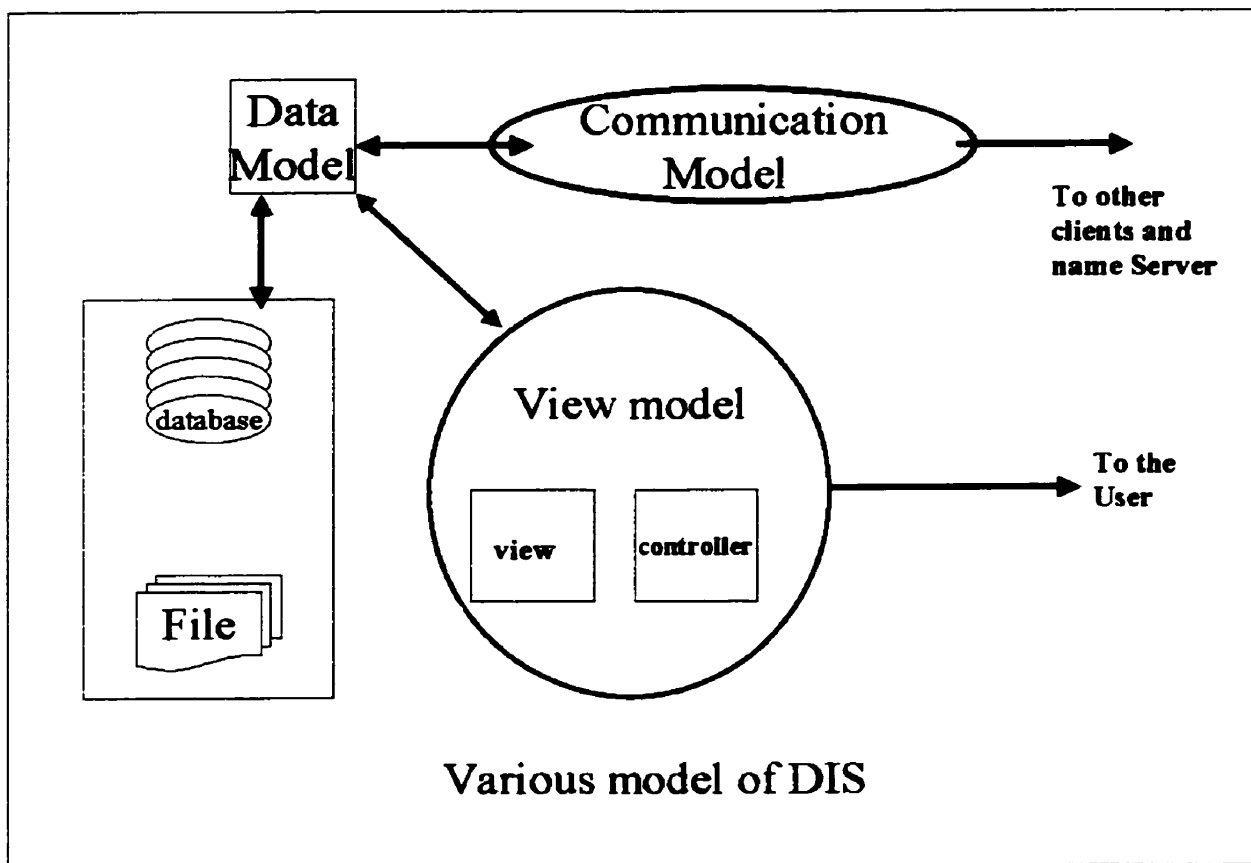


Figure 5.4 – Implementation Model of DIS.

The framework of DIS tries to put the issues discussed in the analysis and design phase into three different models which are the view model, the data model and the communication model (see Figure 5.4).

Here the view model represents the delegate and the data model represents the model of the traditional model-delegate design patterns. The view and data model in DIS help in managing off-line operations whereas the communication model helps in managing on-line operations. Whenever the communication model comes into play we are talking about the distributed object programming where we are calling the remote objects on different virtual machines. These models are separate from each other and it is very easy to change any one of them. This helps to create a core functionality that can be reused by the multiple applications across heterogeneous networks. Each model is further decomposed into the set of related entities that interact with each other to perform the task.

5.3.1 View Model

In the distributed environment of DIS, most of the user's interaction takes place through this model. The view model uses the query methods of the data model to obtain information and displays this information. We can have multiple views of the same data. The view model contains the delegate part of the model-delegate design patterns that is the View and the controller.

The controller object receives mouse clicks or keyboard events from the user. It then translates these events into the manipulator method

which the model understands. In our GUI, views and controllers work very closely together. For example, a controller is responsible for updating a particular parameter in the data model that is then displayed by a view. In some cases a single object function works both as a controller as well as a view. In some situations the controller interacts directly with the view without passing via the model.

GUI is the central class. It listens and handles the interactions from the user along with various GUI classes such as `WorkSpaceGui`, `UserManagementGui`, `SecurityManagementGui`, `NameServerGui`, and `DownloadSpaceGui`. Each of these classes represents a different view of the data model. The GUI's classes have many low-level components added to it which delineate the user with the data from the data model (see Figure 5.5).

Various controllers have been implemented by the GUI's classes and other low-level components using various interfaces such as `TreeExpansionListener`, `MouseListener`, `TreeSelectionListener`, `ActionListener`, etc. Whenever a user presses the mouse or keypad the appropriate listener handles the request and passes the response to the data model.

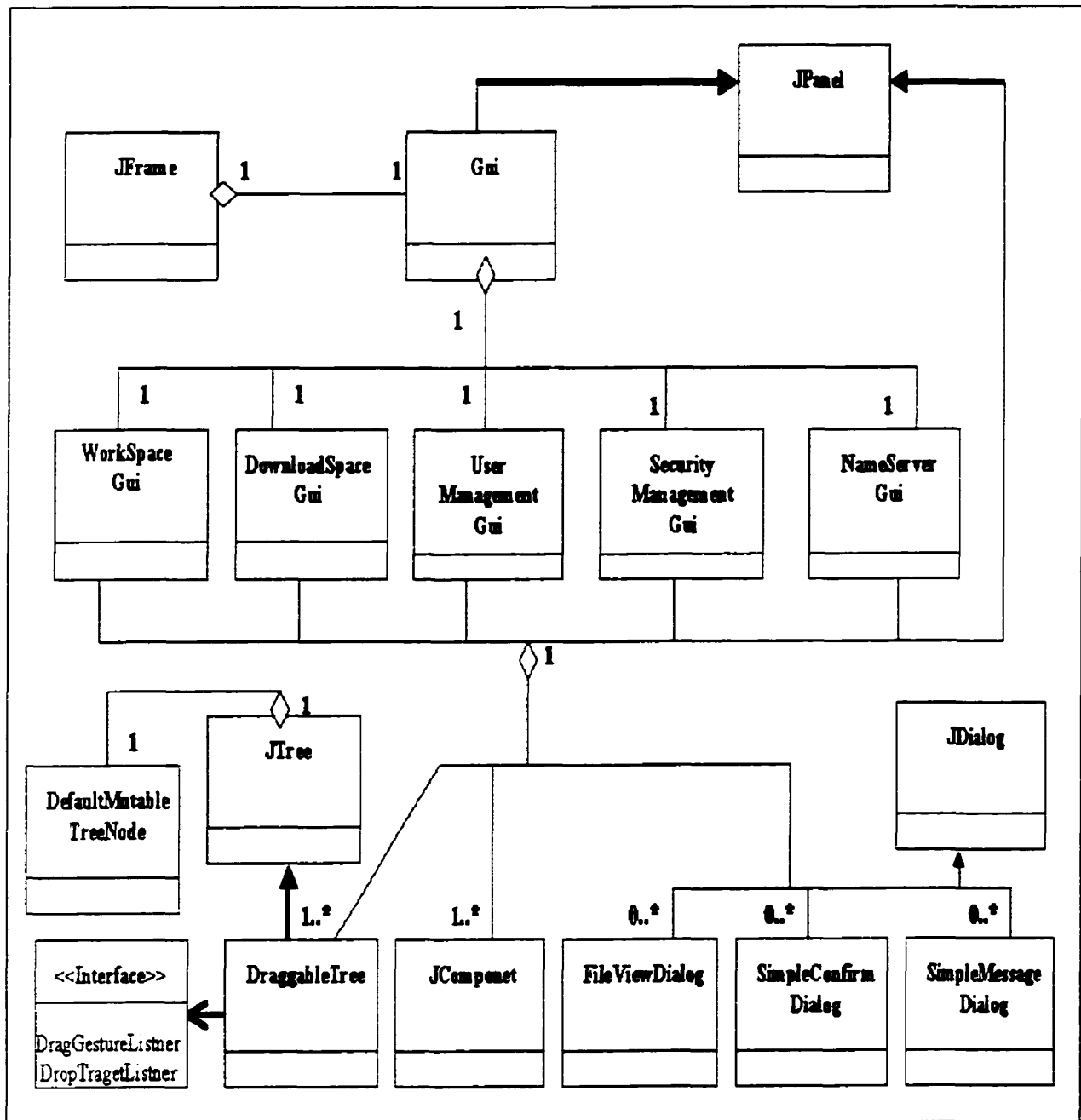


Figure 5.5 – UML of View Model.

5.3.2 Data Model

The data model shows how data is stored in the distributed information system. The basic structure is in the form of the tree. At the top level we have three folders viz. workspace, downloadspace and connected workspace (see Figure 5.6). These folders are like a place holder for the other folders and files.

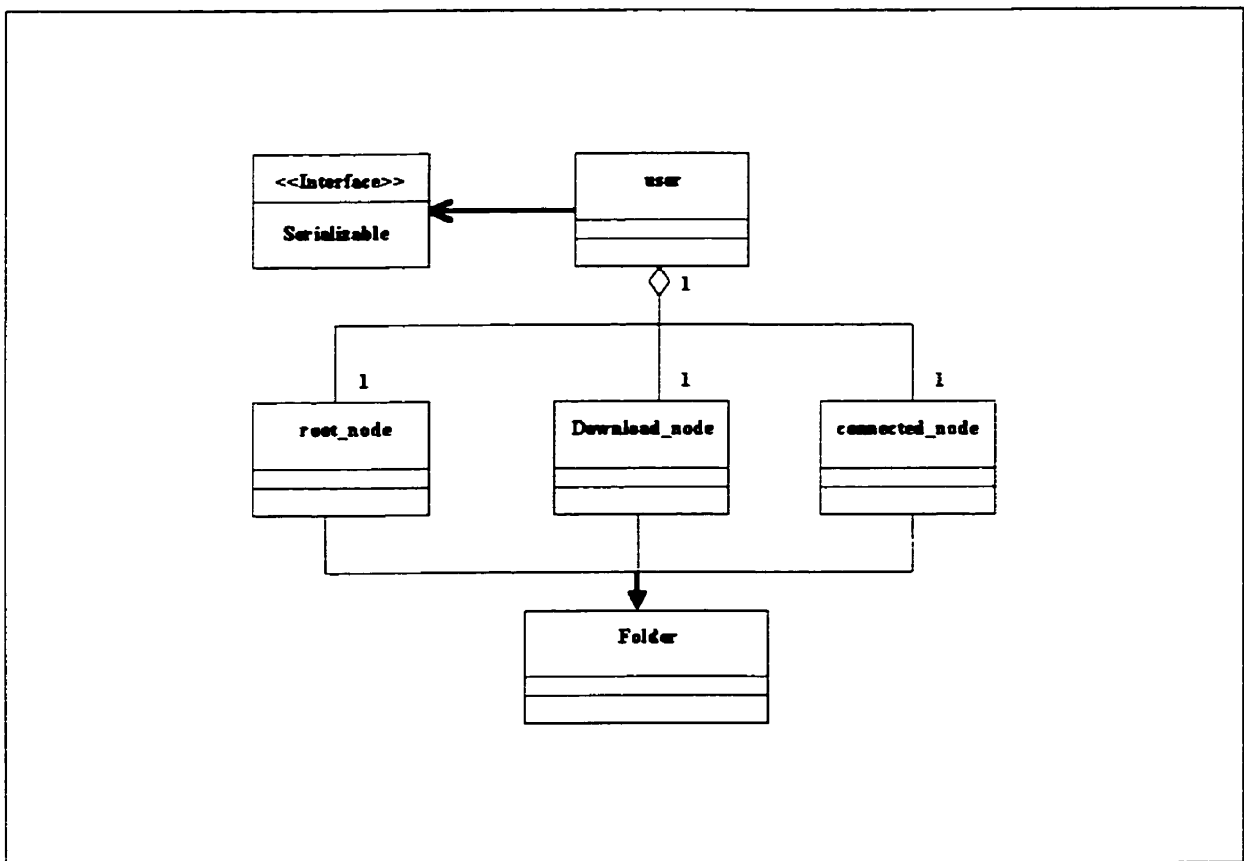


Figure 5.6 – UML of Data Model.

Each folder can be nested further to have folders and files. In the on-line operations of DIS the folders used are the workspace folder and the connected workspace folder. The connected workspace folder acts as

a fetcher and the workspace folder acts like a provider to other clients. You can perform various operations on the folders and files such as deleting, adding and modifying them. These operations are being captured from the user through the view model and then passed to the data model.

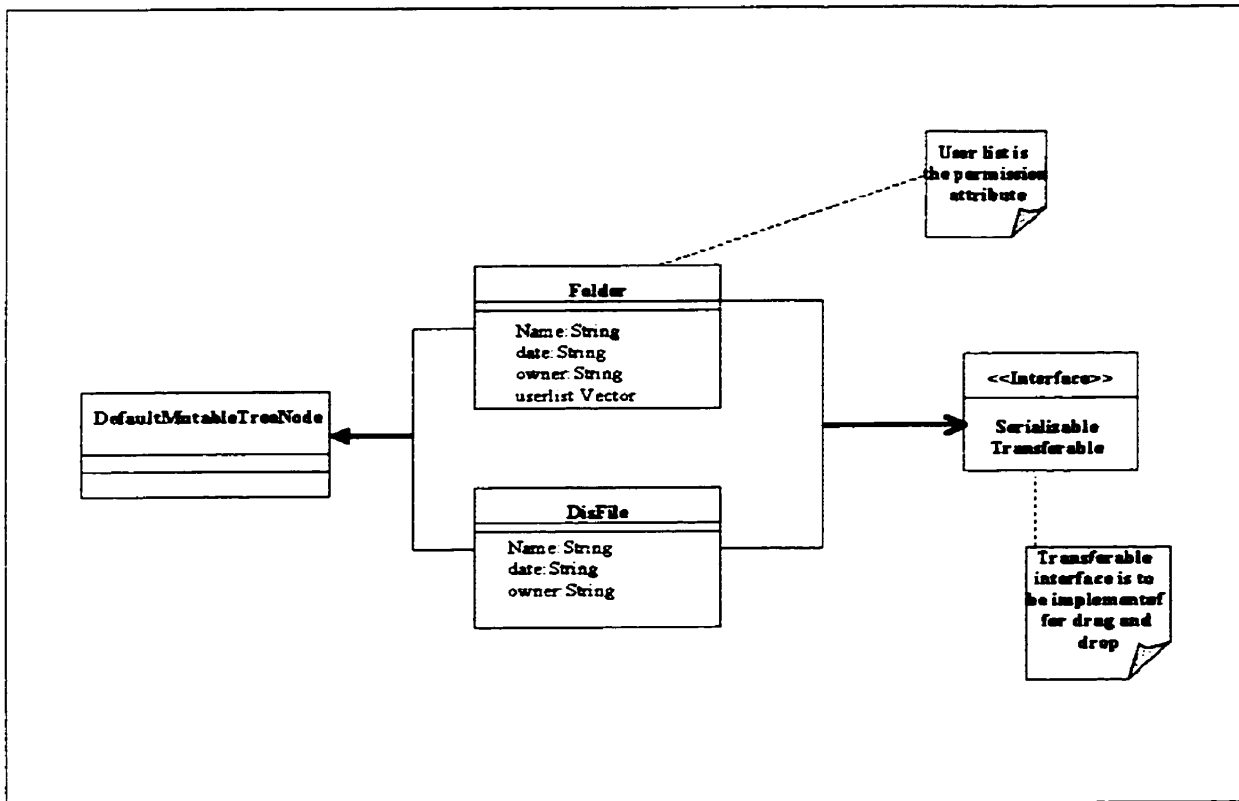


Figure 5.7 – UML of Data Model.

In order to achieve the desired security model discussed in Section 4.5.2 we have a permission attribute attached to each folder. This is a list of all the users that have permission to download the files content of a specific folder (see Figure 5.7).

5.3.3 Communication Model

The communication model comes into play whenever a user desires to perform on-line operations. We divide the communication model into two parts, the name server and LIS server.

5.3.3.1 Name Server

The name server is the standalone server called `ServerAdmin` which runs on a networked computer that has a static IP and provides various services to the user. It implements the interface `iServerAdmin` that provides functionality to the remote users.

5.3.3.1.1 Remote Interface

The remote interface is stored in a separate file called `iServerAdmin`. The remote user objects on a different virtual machine invoke methods from this interface. It supports methods to register a client, remove a client, get the IP address of a user, set the IP address of a user, verify an authentic user, set a password, etc. It also handles various remote exceptions.

The following is the code listed for the `iServerAdmin` interface:

```
public interface iServerAdmin extends java.rmi.Remote{  
void registerClient(String s) throws RemoteException;  
  
void removeClient(String s) throws RemoteException;
```


CHAPTER 5. IMPLEMENTATION OF DIS

```
void setPassword(String userid, String password) throws
RemoteException;
String getIp(String userid) throws RemoteException;
void setIp(String userid , String ip) throws
RemoteException;
Vector getAllUser() throws RemoteException;
Vector getAllActiveUser() throws RemoteException;
boolean validateUser(String userid, String password) throws
RemoteException;
boolean isActive(String userid) throws RemoteException;
}
```

5.3.3.1.2 Implementing the name server

The name server, `ServerAdmin`, is a hashtable of the entry class where each entry has the following attributes: `user_name`, `user_ip`, `user_password` and `active_flag`. The `ServerAdmin` object implements the interface `iServerAdmin` which provides the functionality of retrieving and setting *entry* information from the remote user's objects.

5.3.3.1.3 Starting the name server

The `MainPanel` class provides the basic functionality of the user interface for the name server. `MainPanel` calls the constructor of the server, `ServerAdmin`, to start the server and register itself as a remote object.

The following code shows the initialization of the name server:

```
//Create one or more instances of a remote object
server = new ServerAdmin();

//Loading the variable parameters from the try.ini file such
//as the static IP and the port for the name server so that
//you can change these parameter without compiling the code
//The name server try.ini file look like this
//  {

//      admin = "missys42:1111";

//      variable = "<IP>:<Port number>"

//  };

// load the IP and the port  from the try.ini file into the
// hash table
Hashtable ht = (Hashtable)
ZZParserUtil.convertToDictionaryFromFile("try.ini");

//Register as a remote object
Naming.rebind("//"      +      (String)ht.get("admin")      +
"/ServerAdmin",server);
```

5.3.3.2 LIS Server

As in DIS, each LIS is a client as well as server. So we can have various cases to handle its different scenarios.

5.3.3.2.1 Remote Interface

The LIS server implements the remote interface `iServer` to provide the functionality to the remote users. The remote objects can invoke methods from this interface after being authenticated by the name server.

The following is the code listed for the `iServer` interface:

```
public interface iServer extends java.rmi.Remote{
DefaultMutableTreeNode getTree() throws RemoteException;}

```

5.3.3.2.2 Implementing the LIS Server

The LIS Server implements the interface `iServer` to provide other LIS on different virtual machines with the facilities to download information. The information is serialized and sent to the LIS in the form of a tree structure `DefaultMutableTreeNode`.

5.3.3.2.3 Starting the LIS Server

DIS is the main class that initialized the LIS server. If the "ip" field in the "try.ini" file is "localhost" it gets the local IP from the system that is running the LIS otherwise it will use the value of the "ip" field in "try.ini" file. After that is initialized, the server object registers it as a remote

CHAPTER 5. IMPLEMENTATION OF DIS

object using the IP and the port number from the try.ini file. Once the LIS server is registered to the name server other on-line LIS objects can lookup for this server and invoke methods from the iServer interface.

Try.ini file at the LIS:

```
{ip = "localhost"; <IP address of the LIS server>
port = "2222"; <the port where the client server is running>
admin = "missys42:1111"; < IP of the name server>
user = "vicky"; <user login id> };
```

The following code shows the initialization of the server:

```
//Getting the local IP or user defined IP and port from the
//try.ini file
int i = ((String)Global.hashserver.get ("ip")).compareTo
("localhost");
if (i ==0)
ipaddress = ip.getHostAddress() + ":" + (String)Global.
hashserver.get("port");
else
ipaddress = (String)Global.hashserver.get("ip") + ":" +
(String)Global.hashserver.get("port");
// Starting the server at the local host
Global.my_server = new Server(ipaddress ,user);
//Register as the remote object
Naming.rebind(serverName, this);
```

5.3.3.2.4 Connecting to the name server

Whenever LIS wants to go on-line it has to register itself with the name server. Here we see some of the remote function calls LIS can make to the name server:

```
// Getting the IP address of Name Server from the try.ini
file
String sa = (String)Global.hashserver.get("admin");
String serverName="//" + sa + "/ServerAdmin";
//Look up for the name server remote object
Global.admin_server=(iServerAdmin)Naming.lookup(serverName);
//Setting user IP address and port at the name server. Here
//we are getting the IP from the machine or user defined IP
//and the port from the user try.ini file.
Global.admin_server.setIp(user.user_id, ipaddress);
// Getting all user with-in the system from the name server
Global.all_user = Global.admin_server.getAllUser();
//Getting all the active user from the name server
Global.active_user = Global.admin_server.getAllActiveUser();
```

5.3.3.2.5 Connecting to the other LIS

If you want to connect to LIS other than your own, then the one you wish to connect to must be on-line. First retrieve the IP address of LIS you want to connect to from the name server. Once you retrieve the IP you can then connect to the LIS and retrieve the information tree depending upon your privileges as set by the LIS owner.

```
//Getting the IP from the Name server for the active
//selected user
String t = "/" + (String)Global.admin_server.getIp
((String)list.getSelectedValue()) + "/MyServer";
//Looking for the remote object for the selected user
Global.connected_server = (iServer)Naming.lookup(t);
//Retrieving the information tree from the connected LIS and
//adding to the connected workspace node
connected_node.add(Global.connected_server.getTree());
```

Chapter 6

6 Application of DIS : Distributed Repository of Programming Examples(DRPE)

Example-based learning, see [Neal 89], promotes the idea of using numerous examples to help understand concepts and to move these concepts from short-term memory to long-term memory. Examples are useful if they can be easily browsed and searched and shared by various users. When learning programming, examples are particularly useful because one can always learn from examples of small programs.

Consider, as an example, a specific class for teaching programming in the C programming language, taking place in an electronic classroom. Before the beginning of the class the instructor (provider) makes available examples of programs in C to all students in the classroom. (As

an alternative, the instructor may divide students into groups by giving each group a different example.) Now the students can pull these examples on to their computers. They can view these examples, export them to a favorite compiler, modify them by creating new versions and making these versions available to all students or to just specific students. The entire process can result in a collaborative development of a useful repository of code that can be used not only for learning but also for real every-day programming.

DIS is an ideal candidate to implement Distributed Repository of Programming Examples with a hierarchical tree structure. It has been used to implement DRPEC (**D**istributed **R**epository of **P**rogramming **E**xamples of **'C'**). In this chapter, we illustrate the DIS implemented system in action by studying an example of DRPEC. Throughout the description, we use screenshots to illustrate the interfaces that users will typically encounter while using the system.

6.1 Name Server

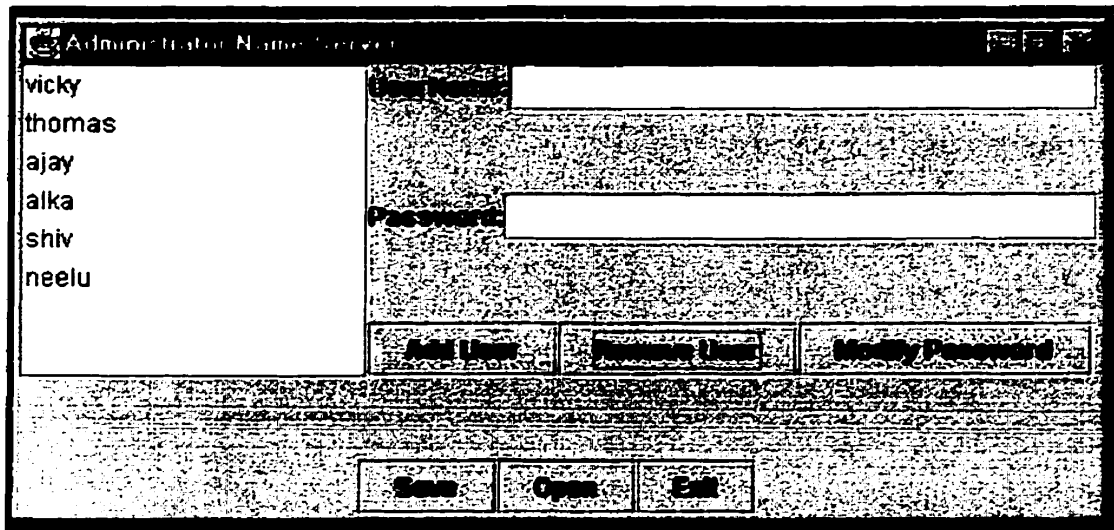


Figure 6.1- Name Server.

The main functionality of the name server from the user's point of view is to handle user requests and from the administrator's point of view is to manage user accounts. Whenever an administrator tries to create a new user account, the system will prompt for the user name to check whether the account already exists or not. It will also check for the blank password. Figure 6.1 shows the name server administrator screen. The administrator uses the same screen to add, modify and delete users.

6.2 DRPEC

6.2.1 Workspace Tab

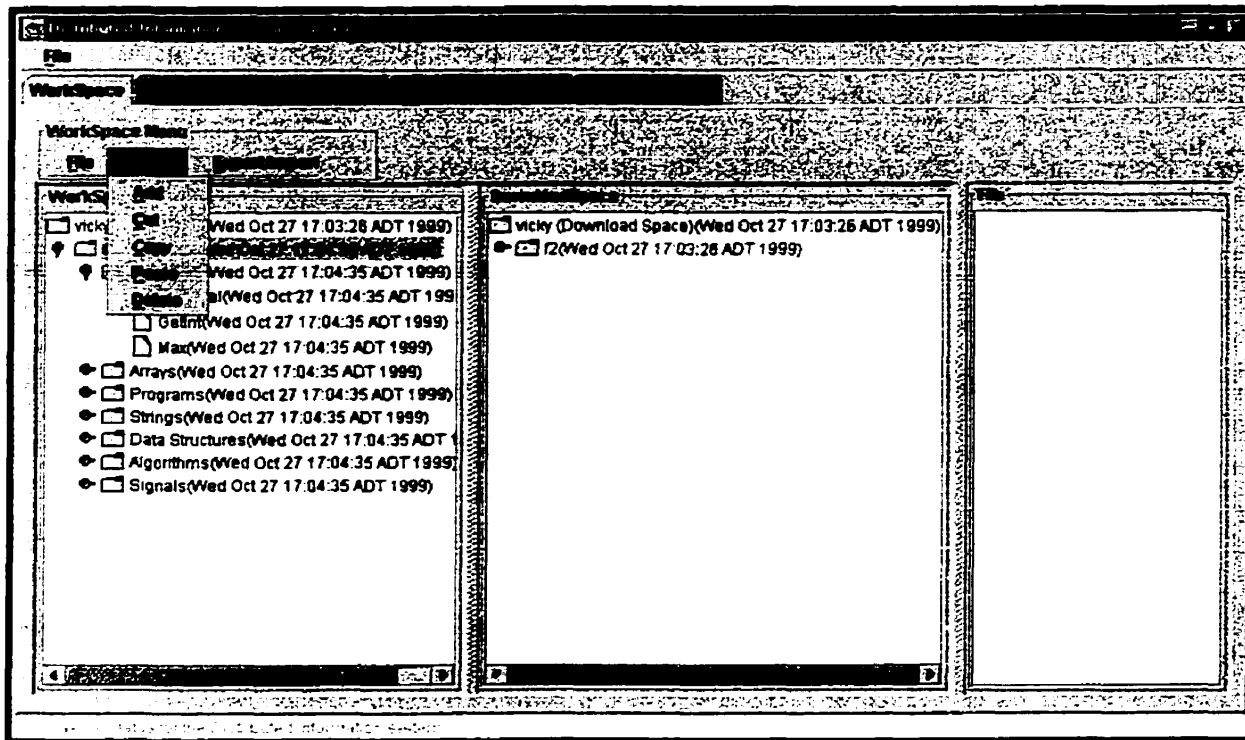


Figure 6.2 – DRPEC Workspace Tab.

As you see in Figure 6.2, the left pane shows the repository of 'C' programs, the middle pane shows the current state of the download space and the right pane shows the content of the file (in case the file is selected). The user can perform various operations such as creating, deleting, moving, and copying classification or document in DRPEC using the workspace tab menu or drag and drop features.

CHAPTER 6. APPLICATION OF DIS:DRPE

The user can import data from a text file into the DRPEC and export DRPEC documents into a text file. These options are available through the workspace tab menu "Export/Import". In order to export a document from the DRPEC, the user has to choose the DRPEC document first and then select the "Export" menu item from the "Export/Import" menu which then prompts for the save dialog box (see Figure 6.3). After this he or she can enter the file name in which the selected document is to be saved.

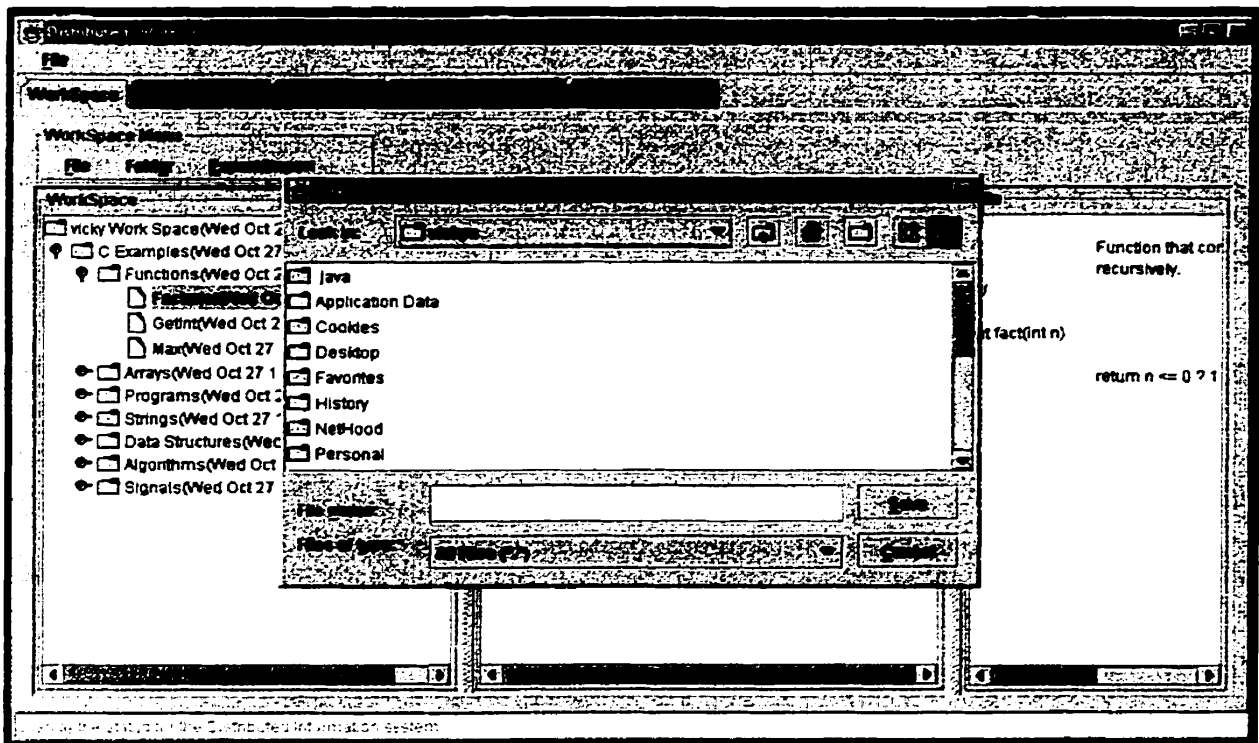
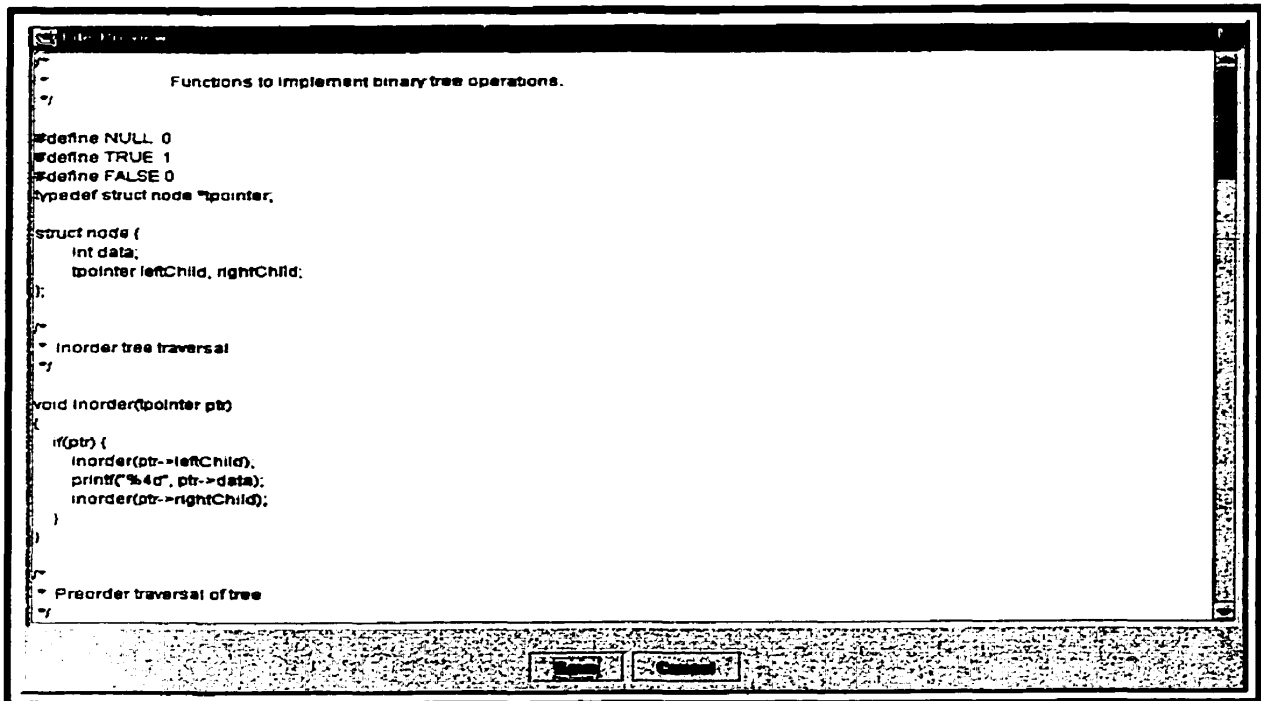


Figure 6.3 – DRPEC workspace (Exporting a file).

In the case of importing a file a prompt for the "open" dialog box will appear where you can select any text file to be imported into DRPEC.

CHAPTER 6. APPLICATION OF DIS:DRPE

The left pane will show the content of the file if the file is selected. If we wish to modify the file just double click it. A file preview dialog will show up where the file can be modified and saved (see Figure 6.4).



```
Functions to implement binary tree operations.
*/
#define NULL 0
#define TRUE 1
#define FALSE 0
typedef struct node *pointer;

struct node {
    int data;
    pointer leftChild, rightChild;
};

/* Inorder tree traversal
*/

void inorder(pointer ptr)
{
    if(ptr) {
        inorder(ptr->leftChild);
        printf("%4d", ptr->data);
        inorder(ptr->rightChild);
    }
}

/* Preorder traversal of tree
*/
```

Figure 6.4– DRPEC File Preview Dialog.

6.2.2 Download Space Tab

Once connected to the other on-line user you can download the classification in your connected workspace pane. All operations in the workspace pane can be performed here using the tab menu. Here user "vicky" has connected to the DRPEC of user "shiv" and has downloaded the classification tree into his connected workspace (see Figure 6.5).

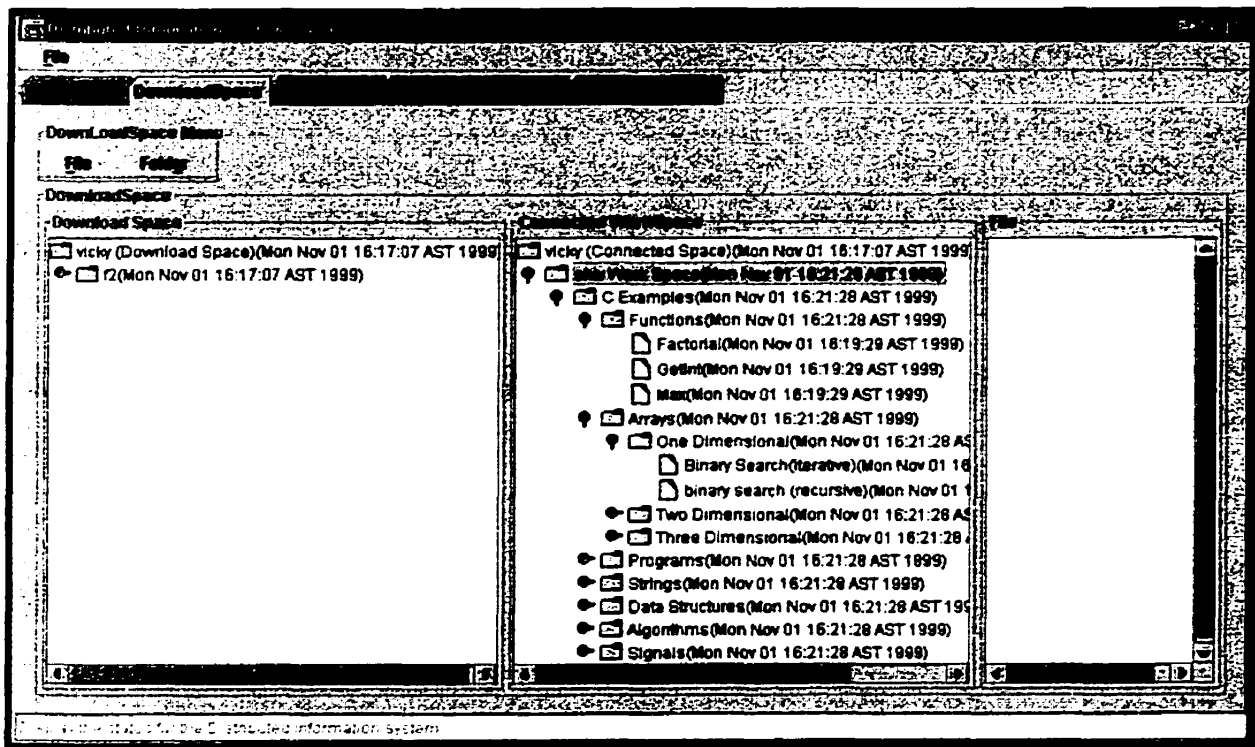


Figure 6.5 – DRPEC Download Space.

6.2.3 Name Server Tab

The DIS user uses the name server tab to connect and talk to the name server (see Figure 6.6). Various operations can be performed here such as logging into the name server, downloading all users, downloading active users, logging off from the name server, changing your password and pinging the name server. Some of these operations require authentication.

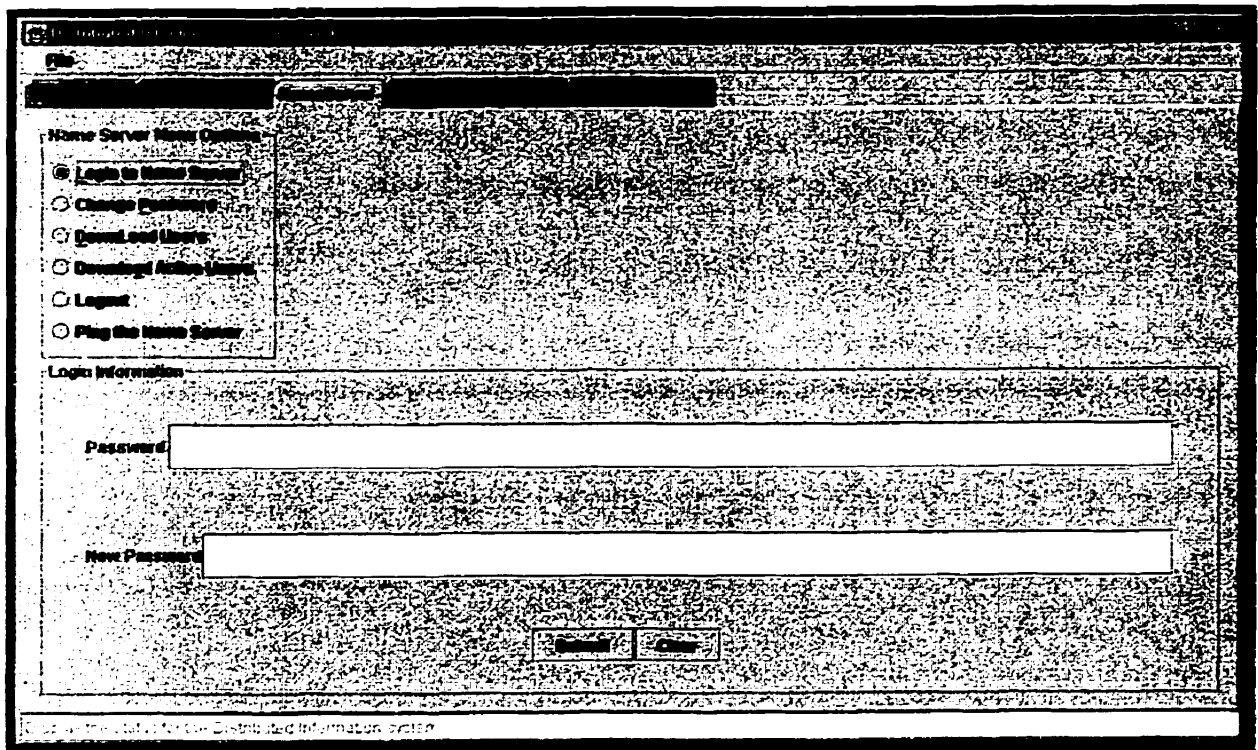


Figure 6.6 – DRPEC Name Server Tab.

6.2.4 Security Management Tab

To manage permissions, the user selects the Security Management Tab (see Figure 6.7). Here, a classification is the smallest unit that the user can grant permissions to. The left pane show the classification tree, the middle pane (User List) shows the list of registered users in the system and the rightmost pane (Selected List) shows the user(s) that can access the selected classification. In order to assign and revoke permissions, the user must first select the classification in the left pane. After selecting the classification the user can add and remove users from the selected list using the bottom pane.

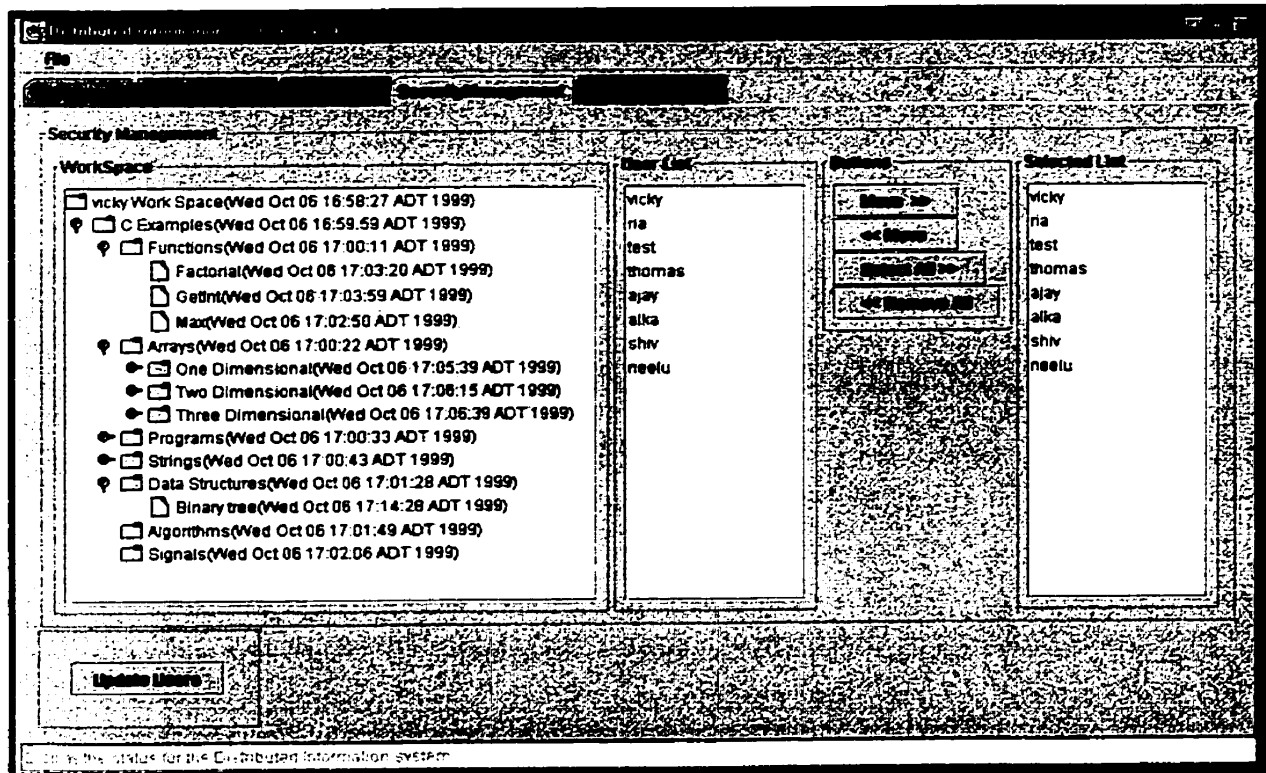


Figure 6.7 – DRPEC Security Management Tab.

6.2.5 User Management Tab

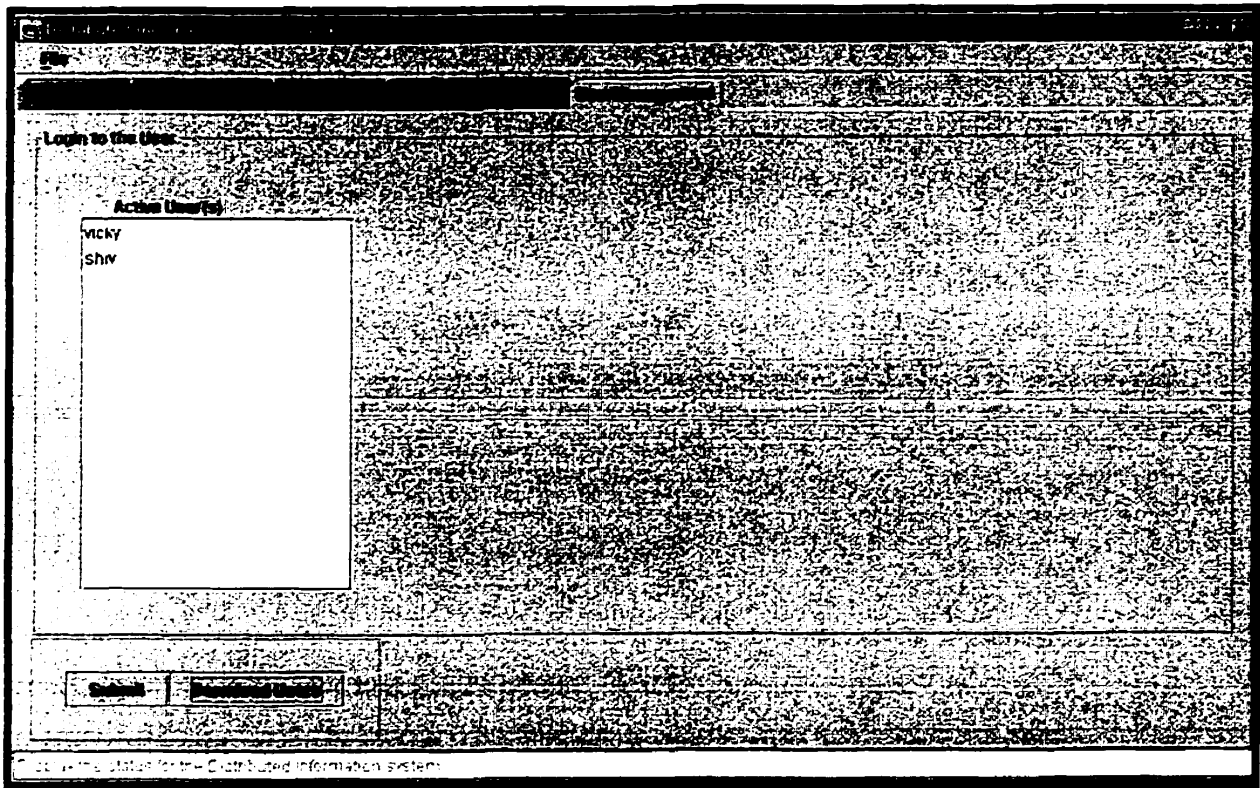


Figure 6.8 - DRPEC User Management Tab.

The list of active users is used to start an interaction with another user. To connect to another user it is simply enough to select this user from the active user list (see Figure 6.8). Note that this connection does not require an authentication because both the users must currently be connected to the name server and therefore have already been authenticated by it. The list of currently active users can be used to select one user and then connect to her or him using the IP address provided by the name server. We can also download the current active user by clicking the "Download Users" button. Once user "vicky" is connected to the another user "shiv", then user "vicky" will be able to

browse user “shiv”’s classification in his or her connected workspace pane (Download Space tab) (see Figure 6.5) depending upon the permission privileges set by user “shiv”.

6.3 Summary

Here are the steps for connecting to user “shiv” workspace from user “vicky”.

Steps for user “shiv”

- Login to the name server (see the name server tab section 6.2.3).
- Set up the “shiv” user repository (see workspace tab section 6.2.1).
- Set up the security model for the “shiv” workspace (see security management tab section 6.2.4).

Step for user “vicky”

- Login to the name server (see the name server tab see section 6.2.3).
- Download the active client from the name server (see name server tab section 6.2.3).
- Select the “shiv” from the list of active client and download the workspace (see the user management tab section 6.2.5).
- View the connected user “shiv” workspace (see the download space tab section 6.2.2).

Chapter 7

7 Conclusions

7.1 Concluding Remarks

Working on this thesis we has experimented with some core areas of object technology such as distributed objects, Java Foundation Classes, and Remote Method Invocation and showed how they work together to build an extensible, portable distributed information system across a heterogeneous network.

Our primary objective is to not only build a system providing distributed data access but to also develop reasonably sophisticated client/server software that does not limit us to specific hardware or

CHAPTER 7. CONCLUSION

operating system. The architecture must be robust, scalable and must be accessible from multi platforms. For example, we wanted to be able to accommodate a user "A" who uses Unix on a PC and a user "B" using a Macintosh. We illustrated that our Java based system used with RMI provides a powerful environment for developing and deploying distributed system over various platforms.

The architecture and model described in this thesis demonstrates the value of layering existing technologies. With very little effort, any other application can benefit from distributed information. We also demonstrated that a distributed information system provides a solid base on which various information services can be built.

We illustrated how DIS provides different services such as acting as both a client and a server. DIS is not only independent of the host architecture and operating system, but it can be deployed to any platform where a Java-compatible VM is available. The power of the Java object model is being utilized to produce a 100% pure Java client/server solution. So we concluded that our approach to building distributed information system is not only practical, but is in some ways superior to the other widely used systems.

7.2 Future Works

We demonstrated that DIS provides a good application to share data over the network. Our future efforts will extend the system to provide more functionality such as sending notifications to users and versioning files. If there is a change in the local information system such as a new file, modified file or a new version of the file, a notification is sent to the notification server for all users who have registered for that notification. The users can login to the notification server and retrieve its notifications. There is a time out for any given notification, after which the notification is deleted from the notification server.

The criteria used for the evaluations include user satisfaction and comparison of the student marks after using DRPE. In our future works, we can use annotations with the classifications and provide groups of users for the security model. We will also implement a more scalable name server so that if one of the name servers is down, the user can connect using the another one.

Currently the system supports point to point connections, but in the future version it will support multi-cast where one LIS should be able to connect to more than one LIS at the same time. The system supports only a pull model in its current version. In the near future, the push model will be incorporated into the system using the notification server.

CHAPTER 7. CONCLUSION

Finally, we would like to indicate that in DIS database side can be re-implemented to use JDBC to connect to desired databases which would not require any changes in either the server or client sides. In the current implementation Drag and Drop is implemented in the first tab; this is related to the problem described in Java Forum [Java 99] where the drag and drop operation if implemented in the tabbed pane works only in the first tab.

Bibliography

- [Booth 81] Booth, Grayce M. *The Distributed System Environment*. New York: McGraw-Hill, 1981
- [Mullender 89] Mullender, Sape. *Distributed Systems*. New York: ACM Press, 1989.
- [Louis 95] Louis [on-line].
Available WWW: <http://www.softis.is> (1995).
- [Eckerson 95] Eckerson, Wayne W. "Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications." *Open Information Systems* 10, 1 (January 1995): 3(20).
- [Microsoft 97] Microsoft (1997). COM/DCOM Specification.
Available WWW: <http://www.microsoft.com/oledev/olecom/title.htm>
- [Orfali 97a] Robert, Orfali; Dan Harky (1997). *Client/Server Programming with JAVA and CORBA*. Wiley & Sons: New York, NY.
- [Rational 98] Rational Software (1998). *The Unified Modeling Language (UML)*.
Available WWW: <http://www.rational.com/>
- [Edelstein 94] Edelstein, Herb. "Unraveling Client/Server Architecture." *DBMS* 7, 5 (May 1994): 34(7).
- [Schussel 96] Schussel, George. *Client/Server Past, Present, and Future* [on-line].
Available WWW: <http://www.dciexpo.com/geos/> (1995).
- [Sun 98] Sun Microsystems. *The Java Language Environment*.
Available WWW: <http://www.javasoft.com/docs/white/langenv/> (11 September 1998).
- [COM 95] "The Component Object Model Specification" (Microsoft Corporation, Digital Equipment Corporation, October 1995)

BIBLIOGRAPHY

- [CORBA 97] "The Common Object Request Broker: Architecture and Specification, Version 2.1" (Object Management Group, et al, August 1997)
- [DCOM 97] Microsoft Corporation. *Distributed Component Object Model Protocol-DCOM/1.0*, draft, November 1996 [on-line]. Available WWW <http://www.microsoft.com/oledev/>
- [OMG 98] Object Management Group home page [on-line]. Available WWW <http://www.omg.org> (1998).
- [Neal 1989] A System for Example-Based Programming. *ACM, Human-Computer Interactions, HCI'89 Proceedings*. pp. 63-67.
- [RMI 97] "Java Remote Method Invocation" (Sun Microsystems, Inc., December 1997)
- [RMI 99a] RMI and Java Distributed Computing white paper by Doug Sutherland System Architect JavaSoft™, A Business Unit of Sun Microsystems, Inc
- [RMI 99b] Java remote method invocation - distributed computing for Java , White paper <http://java.sun.com/marketing/collateral/javarmi.html>
- [Muldner, Shiv 00] Distributed Repository of Programming Examples accepted for EDMEDIA 2000, Montreal, Canada June 2000.
- [Java 99] Problem implementing D&D in Tabbed Pane Available WWW: <http://forum.java.sun.com/forum?14@@.eeec519>
- [Ulf, Raymond 97] Ulf Hermjakob, Raymond J. Mooney: Learning Parseand Translation Decisions from Examples with Rich Context. *ACL 1997*: 482-489
- [Adler 95] Adler, R. M. "Distributed Coordination Models for Client/Sever Computing." *Computer* 28, 4 (April 1995): 14-22.

BIBLIOGRAPHY

- [Benda 97] **Miroslav Benda: Architecture Perspective:
Middleware: Any Client, Any Server. IEEE Internet
Computing 1(4): 94-96 (1997)**

Appendix A

DIS Installation Guide

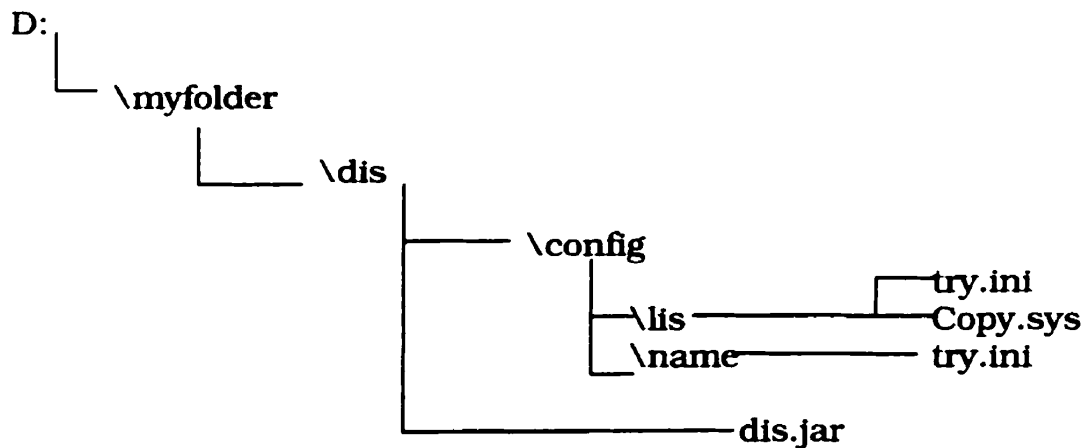
In this Appendix, we describe how to install and run the DIS name server and client (LIS) software. Both name server and client installations need Java Development Kit (JDK) version 1.2.

I) Software Installation

Here are the steps to install the DIS name server and make available to clients:

STEP 1 Download the zip file "dis.zip" and unzip the "dis.zip" file in the desired destination directory ("D:\myfolder" for example).

STEP 2 After unzipping to the "myfolder" it will create the following directory structure:



II) Running LIS On-line

Before you can run LIS you must have a user account on the name server and you must know the static IP and port where the name server is running. Here are following steps to install and run the LIS:

STEP 1 Set the Java classpath to the "dis.jar" file and to your current directory.

Format

```
set classpath=<Path name for the dis.jar file>.;.
```

Actual Command

```
set classpath=d:\dis\myfolder\dis.jar;..;
```

STEP 2 Open the try.ini file in the "lis" directory and enter the IP and the port number of the LIS server, name server and your user id in the format given below:

Format

```
{  
  
ip = "<IP address of the LIS server or  
"localhost" if you wish to grab the IP from the  
system>";  
  
port = "<the port where the LIS server is  
running>";  
  
admin = "< IP name server>:<portname server>";
```

Appendix A

```
user = "<user login id>";  
  
};
```

Actual File

```
{  
  
ip = "localhost";  
  
port = "2222";  
  
admin = "122.3.4.2:1111";  
  
user = "vicky";  
  
};
```

- STEP 3** Start the RMI registry for the LIS server at the desired port ("2222" for example) by executing the following command:

```
start rmiregistry 2222
```

- STEP 4** Run the following command to start the LIS.

Format (all one line)

```
java -classpath <jar file path name>  
lis.viewpackage.Gui <zip folder path name or  
nothing if the installation is on "C:" Drive>
```

Actual command (all one line and use slashes as shown)

```
java -classpath d:\myfolder\dis\dis.jar  
lis.viewpackage.Gui D:/myfolder/dis/
```

III) Running LIS Off-line

STEP 1 Run the following command to start the LIS.

Format (all one line)

```
java -classpath <jar file path name>  
lis.viewpackage.Gui <zip folder path name>
```

Actual command (all one line and use slashes as shown)

```
java      -classpath      d:\myfolder\dis\dis.jar  
lis.viewpackage.Gui D:/myfolder/dis/
```

IV) Running the Name Server

STEP 1 Set the Java classpath to the "dis.jar" file and to your current directory.

Format

```
set classpath=<Path name for the dis.jar file>::;
```

Actual Command

```
set classpath=d:\dis\myfolder\dis.jar;;
```

STEP 2 Open the try.ini file in the "name" directory and enter the static IP and the port number where you wish to run the name server:

Format

```
{  
admin = "<IP>:<Port number>";  
};
```

Actual File

Appendix A

```
{  
admin = "122.3.4.2:1111";  
};
```

STEP 3 Start the RMI registry for the name server at the desired port ("1111" for example) by executing the following command:

```
start rmiregistry 1111
```

STEP 4 Run the following command to start the name server.

Format (all one line)

```
java -classpath <jar file path name>  
name.viewpackage.MainPanel <zip folder path name>
```

Actual command (all one line and use slashes as shown)

```
java -classpath d:\myfolder\dis\dis.jar  
name.viewpackage.MainPanel D:/myfolder/dis/
```