Technical report, IDE1067, November 2010

Architecture of a Distributed LDAP Directory

Master’s Thesis in Computer Systems Engineering

Nuruzzaman Bepari, Mohammad Zabedul Alam & Md. Syadur Rahman

School of Information Science, Computer and Electrical Engineering
Halmstad University
Figure on cover page: Different components of an LDAP directory services from page no. 18, Figure 2.1: Components of Directory system.
Acknowledgements

Master’s thesis is the part of Master’s Program for the fulfillment of the Master’s Degree specialization in Computer Systems Engineering at Halmstad University.

For completion of the thesis, we are thankful and glad to our supervisor, Tony Larsson, Professor of Embedded Systems, School of Information Science, Computer and Electrical Engineering, Halmstad University, for his cordial supervision and guidance throughout this thesis. Moreover, we are thankful to the library personnel of Halmstad University who helped us with collecting the necessary papers.

In addition, we owe gratitude to some of our friends of Halmstad University, who shared their ideas with us.

Nuruzzaman Bepari, Mohammad Zabedul Alam & Md. Syadur Rahman
Halmstad University, November 2010
Abstract

LDAP directory servers are a kind of hierarchical databases aimed for indexed information about a given domain that is very useful for network administration, especially for databases containing information about network users and network devices. In big organizations in which there are a great number of users and a network is divided into several sub-domains, the performance of an LDAP directory may suffer from degradation in moments of high pick usage, e.g. beginning of a work day. A very straightforward way to solve this problem is to run the service using more powerful computers. However, this is not a scalable solution, as the size and the dimension of the LDAP directories may grow exponentially if additional concepts are added to it. Therefore, a smart solution is the architecture of a distributed LDAP directory, in which a central LDAP server just maintains pointers to the other LDAP servers, which provides information about the considered sub-domains of the company with different kinds of information, specifically network users, network devices, customers, products, role tasks and sensitive information. In our thesis, we have proposed such an architecture of a distributed LDAP directory based library system, in which there will be a central administrator of a book library who will manage and share data among other subordinate and local libraries, having a direct access right to the LDAP directories handling the subordinate libraries.
Table of Contents

1 INTRODUCTION ......................................................................................................................... 1

1.1 APPLICATION AREA AND MOTIVATION ............................................................................. 1

1.2 PROBLEM STUDIED ............................................................................................................... 1

1.3 APPROACH CHOSEN TO SOLVE THE PROBLEM ................................................................. 1

1.4 THESIS GOALS AND EXPECTED RESULTS ........................................................................ 2

2 STRUCTURE OF AN LDAP DIRECTORY .................................................................................. 3

2.1 Working Procedure of an LDAP Directory ............................................................................ 5

2.2 The LDAP Operation Models .............................................................................................. 6

2.2.1 The LDAP Information Model ....................................................................................... 6

2.2.2 The LDAP Naming Model ............................................................................................. 7

2.2.3 The LDAP Functional Model .......................................................................................... 8

2.2.3.1 The LDAP Interrogation Operations .......................................................................... 8

2.2.3.2 The LDAP Update Operations ................................................................................. 11

2.2.3.3 The LDAP Authentication and Control Operations .................................................. 13

2.2.4 The LDAP Security and Access Control Model ............................................................ 14

2.2.4.1 Security ................................................................................................................... 14

2.2.4.2 Authentication ......................................................................................................... 15

2.2.4.3 Access Control ......................................................................................................... 17

2.2.5 LDAP APIs .................................................................................................................... 17

3 DESIGN OF AN LDAP DIRECTORY .................................................................................... 19

3.1 DATA DESIGN ...................................................................................................................... 19

3.1.1 Data Elements Characteristics ....................................................................................... 20

3.1.2 Data Sources .................................................................................................................. 21

3.1.3 Managing Good Relationships with other Data Sources .............................................. 22

3.1.4 Synchronization ............................................................................................................. 22

3.1.5 Batch Updates ............................................................................................................... 22

3.2 SCHEMA DESIGN .............................................................................................................. 22

3.2.1 DIT Layout .................................................................................................................... 22

3.2.1.1 Top Levels ............................................................................................................... 22

3.2.1.2 Lower Levels ........................................................................................................... 23

3.2.2 Elements of LDAP Schemas ....................................................................................... 24

3.2.2.1 Attributes ............................................................................................................... 24

3.2.2.1.1 Naming attributes .............................................................................................. 24
Introduction

1 Introduction

In our everyday lives, we are using different kinds of directories for many different purposes. A directory is a collection of data or list of data. For example, phone books and yellow pages, TV guides, shopping catalogs and library catalogs are directories. Unlike RDMS, directories show data hierarchy. The use of directories is increasing rapidly, day by day. So, well designed techniques and protocols are crucial for directory applications.

1.1 Application Area and Motivation

The LDAP (Light Weight Directory Access Protocol) is a standard and extensible directory access protocol [1]. It is used by LDAP clients and servers in order to communicate with each other. It provides easy access to directories.

This thesis proposes the architecture of a distributed LDAP directory implementing a book library system. In each library, there will be users, administrators, librarians, different kinds of books and existing services. In this library system, there is a central library. Under the central library, there are subordinate libraries. Each library contains an LDAP directory. In each directory, different kinds of information about administration, librarians, users, books and services are stored. In each library based on the LDAP directory, the users will be able to search the books. However, they will not be able to modify the data. They can communicate with the administrator of their belonging library when needed. For example, when they can not find the books that are needed, they can communicate with the administrator so that the administrator can manage the books for them from somewhere else. However, the users of a library can not communicate with the administrators of other libraries. The librarians will be able to update the directory data. On the other hand, the administrator will get all rights reserved in the directory. He will be able to update and manage the data. He will also be able to control the user’s access. Moreover, he will be able to communicate with the administrators of other libraries when needed. The central administrator of central library controls and manages data of other libraries.

1.2 Problem Studied

Many applications use the LDAP directories to manage the data centrally. However, to control the data centrally is not always a good approach. This is because, when this process is used in a big company in which there are hundreds of thousands of entries, the performance of this kind of solution may suffer from degradation in moments of high pick usage, e.g. the beginning of a working day. Just deploying powerful computers does not solve this problem. This problem can be solved if one distributes the data among many LDAP servers by a central server. In this thesis, the aim is to implement and then evaluate scalability and reliability aspects of a library system based on the LDAP directory in which there will be a central library that will be able to share and distribute the data.

1.3 Approach Chosen to Solve the Problem

We have implemented our thesis work using LDAP directories and Apache Directory Server because the LDAP operations are easy and fast. Using LDAP server, data can be managed,
Introduction

updated and searched quickly. It may also be possible to implement this work using RDMS (Relational Database Management System) such as for example Oracle. The RDMS shows data as tables (rows and columns) in which one table relates with others by the matching entries. On the other hand, a LDAP directory shows data hierarchically. Hence, it takes much effort and time in order to perform the search, update and modify operations in RDMS.

The LDAP directory, for the book library is implemented as an application of the Apache Directory Studio and the Apache Directory Server V1.5 that supports LDAP v3. The Apache Directory Server is an open source program. It is an embeddable directory server entirely written in Java. It has been certified the LDAPv3 compatible by the Open Group. Also, it provides strong security protocols, namely SSL and TLS. For the communication between a user and an administrator or between the administrators of different libraries, we have done client/server socket programming written in Java programming language using Eclipse platform.

Moreover, Jxplorer was used as LDAP client software. Using this software, the clients can search the directory data. The JXplorer [11] is an open source LDAP browser originally developed by “Computer Associates' eTrust Directory Development Lab”. It is a standard compliant general purpose LDAP browser that can be used to read and search any LDAP directory, or any X500 directory with an LDAP interface.

The thesis report is organized into several chapters. In chapter 2, we have clearly discussed the structure of the LDAP directory. Here, the description of the directory and the LDAP has been provided. Also, various LDAP models have been described. In chapter 3, we have presented the description of the LDAP directory design. Clear descriptions of the data design, the schema design, the namespace design, the topology design and the replication has been given. In chapter 4, we have mentioned the problem that is implemented in our thesis work. The solution of this problem is provided with details in chapter 5. In chapter 6, experimental results are provided. In chapter 7, the conclusions and future work are given. After this, references are provided and then, at the end of the report, the Appendix is provided in which the client/server programs have been presented.

1.4 Thesis Goals and Expected Results

The goal of this thesis is to give the architecture of a distributed LDAP directory. This thesis work will be implemented on a library system based on the distributed LDAP directory. In this library system, there will be a central library and many other local and subordinate libraries. The central library will control and distribute data of all local and subordinate libraries.

In the experiments, an administrator of a library will be able to add, delete, modify, control and manage the data of a directory from a remote network place using LAN (Local Area Network). The users of a library will be able to search books and existing services using LAN. The librarians will be able to modify data. Moreover, the central administrator of the central library will be responsible for the distribution of data among all subordinate and local libraries, and the access control of users.
2 Structure of an LDAP Directory

A directory is a collection of data or list of data. In our everyday lives, we use various directories for many purposes. For example, phone books and yellow pages, TV guides, shopping catalogs and library catalogs are directories. There are different kinds of directories. However, we can classify them into two categories, namely offline directories and online directories.

**Offline Directory:** The directories that we use in our everyday lives for finding things like phone books and yellow pages, TV guides, shopping catalogs and library catalogs are offline directories.

**Online directory:** directories used in the computer and network world are similar in many ways in real world directories with some significant differences. Online directories may differ from offline directories in the following ways:

**Dynamic:** offline directories are relatively static because data in these directories hardly change. For instance, the phone book comes once a year and a new TV guide is produced every week. By contrast, in an online directory, data can be updated frequently. An authorized administrator can, automatically, update directory data from a remote site, although the directory data are normally kept and not frequently changed.

**Flexible:** online directories are more flexible than offline directories in the types of information they can store and in the way that information can be managed and searched. An offline directory contains static contents which can not be easily extended by new data. Moreover, to get the desired information is cumbersome. However, in an online directory, the contents can be easily extended by new data and both searching and managing information is flexible.

**Secure:** a printed directory provides almost no security if in wrong hands. On the other hand, an online directory needs to provide more security. The users’ access is controlled and only the authenticated users can access the directory. The authorized administrators can modify, update and manage directory data.

**Directory services:** a directory service is comprised of software, hardware, policies and plans, and administrative procedures. These things are vital to make the directory services and information available to the clients. A directory service will have at least the components, namely the information contained in the directory, software servers, software clients, hardware, supporting operating systems, network infrastructure, policies, procedures and software used to maintain the clients and servers.
LDAP is a standard, extensible directory access protocol—a common language that LDAP clients and servers use to communicate with each other [1]. Unlike a "heavyweight" protocol, such as the X.500 Directory Access Protocol (DAP), LDAP is a lightweight protocol. “Lightweight” means that it is easy to implement, efficient and straight-forward, although it provides high functionalities. X.500 DAP uses a complex encoding method and OSI network protocol stack, but LDAP uses simplified encoding methods and runs on the top of the TCP/IP layer. LDAP promises the following functions for different kinds of professionals [1]:

- LDAP helps a system administrator to centrally manage users, groups, devices, and other data.
- LDAP helps an IT decision maker to avoid a single vendor and/or operating system platform.
- LDAP helps an IT decision maker to reduce the total cost of ownership by reducing the number of distinct directories.
- LDAP helps a software developer not to be bound to a single vendor and/or operating system platform.
- LDAP helps a software developer to save development time by avoiding the need to construct their own user and group management database.
2.1 Working Procedure of an LDAP Directory

LDAP is a client/server based protocol where the clients send requests to the server while the server processes client requests and sends the results back to the clients. For instance, Hypertext Transfer Protocol (HTTP) typically used to serve Web pages; and Internet Message Access Protocol (IMAP) used to access electronic mail messages are client/server protocols. The LDAP server computer might have a lot of RAM, fast disks and fast processors in order to process client requests quickly and properly because many clients can send requests to the server at any one time. The LDAP client computer, however, does not need to have highly configured computer. LDAP client and server communicate with each other by passing messages. The client creates a message containing its requests and sends it to the server. The server receives these messages and processes them. Then, the server sends the results back to the client as a series of LDAP messages.

For example, if an LDAP client wants to search a specific entry in a LDAP directory, first he will create an LDAP message with a search request and then, he will send it to the server. The LDAP server will receive that message and retrieve the entry from its database. Finally, the server will send a message to the client containing the entry result. It will also send a result code to the client.

Fig. 2.2.1 (a). A client retrieves a single entry from the directory.

However, if a client sends a message to the LDAP server for searching a specific entry, the LDAP server will check whether multiple copies of that entry are available in the directory or not. If the server finds that multiple copies of an entry are available, it will send all copies of entries as a sequence of LDAP messages, one for each entry. Finally, it will also send a result code message to terminate all results.

Fig. 2.2.1(b). A client searches the directory, and multiple entries are returned.
Structure of an LDAP Directory

Moreover, LDAP protocol allows the client to send multiple search requests at a time. For instance, a client can, simultaneously, send two messages with the search request to the server, generating a unique ID for each message. Then, the server will receive these messages and will process the results. After processing the results, the server will return the entries successively on the basis of a message ID.

2.2 The LDAP Operation Models

LDAP defines four basic models that fully describe its operation, what data that can be stored in LDAP directories and what can be done with that data. These models are described in the following sections.

2.2.1 The LDAP Information Model

The LDAP information model describes how we can define the types of data to be stored in a directory. Actually, it depicts the basic building blocks of the LDAP directory. An entry is a collection of information about an object. It is the basic unit of information in a directory. Information is stored in a directory as entries. A typical directory contains thousands of entries corresponding to people, departments, servers, printers, hardware, and other real-world objects in the organization. The figure represents a portion of a typical directory containing some real-world objects in an organization.
Structure of an LDAP Directory

An entry is comprised of a number of attributes where each attribute contains a type and one or more value pairs. The type describes what kind of information that will be put in the attribute while the value contains actual data. As an example, the following figure shows an entry with different attribute types and values. This entry contains a person’s common name (cn), surname (sn), telephone number and mail address.

Moreover, attribute types contain syntax such as caseIgnoreString or caseExactString. Syntax is important while comparing attribute values at the time of searching. If the caseIgnoreString syntax is defined, the strings will be ordered lexicographically and the case is not significant. For instance, the attribute values Smith and smith will be considered the same when searching or comparing values if the caseIgnoreString syntax is defined. However, the caseExactString syntax specifies that case is significant when searching or comparing attribute values. For example, the values Smith and smith will not be considered the same for the caseExactString syntax.

Attributes are classified into two categories, namely user and operational. User attributes are normal attributes which can be modified by any user with appropriate permissions. By contrast, operational attributes are special attributes that modify the operation of a directory server. For example, modifytimestamp is an operational attribute that reflects the last modified time of an entry. Attribute value can be single or multiple. For example, the attribute type country name(c) only has single value. However, the attribute type description has single or multiple values.

2.2.2 The LDAP Naming Model

To organize a directory’s data is a difficult task for a developer. The LDAP naming model enables a developer to organize entries properly in a part of LDAP directory. Managing directory entries properly also enables a developer to enjoy ease of searching, modifying and updating specific information. Actually, the performance of operation in a LDAP directory depends on the design and organization of its data. For example, if we keep our books at sixes and sevens in the table, it will be cumbersome to find a specific book and also the table will
look bad. In an LDAP directory, data is organized hierarchically, and the entries are arranged in an inverted tree structure, as shown in fig. 2.2.2.2 (a).

Fig. 2.2.2.2 (a). A directory tree.

2.2.3 The LDAP Functional Model
The LDAP functional model describes the basic operations performed on the LDAP directory to access data. There are nine common operations performed on the data stored in the directory tree which can, finally, be categorized into three groups, namely interrogation operations, update operations and authentication and control operations. Each group of operations is described below.

2.2.3.1 The LDAP Interrogation Operations
There are two LDAP interrogation operations, namely the search and the retrieve operation. The search operation helps the LDAP clients to find a specific entry stored in the directory. As there is no LDAP read operation, a specific form of search operation is needed. To perform a search operation, eight parameters are needed. The base object, expressed as distinguished name (DN), is the first parameter that denotes the top of the tree. The second parameter is scope, which is categorized into three types, namely subtree, onlevel and base. A scope of subtree is used to search the entire tree from the base object to the leaves of the tree. A scope of onlevel is used to search the immediate children of the entry at the top. A scope of base is used to retrieve a particular entry the directory. The three types of scope are depicted by the fig. 2.2.2.3.1.
The third search parameter is deferAliases. This parameter is used in order to dereference aliases while performing a search operation. There are four values for this parameter which are given below:

- neverDerefAliases- never dereferences aliases while searching.
- derefInSearching- dereferences aliases while searching.
- derefFindingBaseObject- dereference aliases in locating the base object of the search, but not for searching subordinates.
- derefAlways- dereferences aliases both in searching and in locating the base object.

The fourth search parameter is the size limit, which enables an LDAP client to retrieve a certain number of entries. For instance, if a client passes a size limit of 50, but the LDAP server finds 500 matching entries, the server will send only the first 50 matching entries, with the result code LDAP_SIZE_EXCEEDED sent to the clients. If the client wants to retrieve all the matching entries, a size limit of 0 should be defined.

The fifth search parameter is the time limit, which allows a client to elapse the maximum time for a search request. For example, when a client sends a search request with a time limit of
Structure of an LDAP Directory

two seconds, the server will stop processing if two seconds is exceeded and send a result code LDAP_TIME_EXCEEDED to the client. A time limit of zero indicates that no time limit will affect the searching operation.

The sixth search parameter, a Boolean parameter, is the attrsOnly, which is used by a client if he wants to find out which attributes’ types are possessed in an entry hiding their actual values. The server will send only the attribute type but not the attribute value when attrsOnly is set true. When it is set false, the server will send both attribute types and their corresponding values. Various types of search filters are described below:

The sixth search parameter, the attrsOnly parameter, is a Boolean parameter. If it is set to true, the server will send only the attribute types to the client: attribute values will not be sent. This can be used if the client is interested in finding out which attributes that are contained in an entry, but not in receiving the actual values. If this parameter is set to false, attribute types and values are returned. The eighth and final search parameter is a list of attributes to be returned for each matching entry. The seventh search parameter, a Boolean combination of attribute-value assertions, is the search filter that describes what types of entries will be returned. An attribute-value assertion has two parts, namely an attribute name and a value assertion. There are different kinds of search filters which are described below:

**Equality filter:** an equality filter allows a client to search an entry that matches with an exact value. For instance, if a client searches an entry with sn = Nayan, the server will send entries that exactly match with sn = Nayan. Here, sn is the attribute type and Nayan is the attribute value.

**Substring filter:** a substring filter is used to retrieve entries that match an attribute value that starts with a substring of the value. For example, if a client searches entries with substring sn = nayan*, the server will send all entries that start with sn = nayan. The filter (sn = nay*ar) will match entries where the surname starts with nay and ends with ar, specifically nayanathar. The filter (sn = *yan) will match entries where the surname ends with yan like nayan. The filter (sn = *yan*) will match entries that contain a string yan like nayanath.

**Approximate Filters**

An approximate filter is used to match entries in which the attribute type has near attribute value. For example, the filter (sn~ = jensen) will return entries in which the surname attribute has a value that sounds like jensen.

"Greater Than or Equal To" Filters

LDAP server supports "greater than or equal to" filter to match entries in which the attribute type has value that is greater than, or equal to, the searching value. For example, the filter (sn>= nayan) will return all entries where the surname is greater than, or equal to, nayan lexicographically.

"Less Than or Equal To" Filters

An LDAP server also supports a "less than or equal to" filter to match entries in which the attribute type has value that is less than, or equal to, the searching value. For example, the
Structure of an LDAP Directory

filter(sn<=nayan) will return all entries where the surname is less than, or equal to, nayan lexicographically.

**Presence Filters**

A presence filter is used to match any entry that has at least one value for the attribute. For example, the filter (telephoneNumber=*) will return all entries that have a telephone number.

**Extensible Match Filter**

The last type of search filter is the extensible match filter. Only LDAPv3 support this filter. The last type of search filter is the extensible match filter. It is only supported by LDAPv3 servers. Its purpose is to allow the implementation of new matching rules for client and server.

**Combining Filter Terms**

AND and OR operators are used in order to combine filters. The AND operator is denoted by an ampersand (&) symbol, while the OR operator is denoted by the vertical bar (|) symbol. These operators are used before the search filter. For example, the filter (&(sn=javed)(l=Mountain View)) will match all entries in which the surname is javed and that also have an l (locality) attribute of Mountain View. On the other hand, the filter (|(sn=nayan)(sn=javed)) will match all entries with a surname of nayan or javed. Moreover, the filter (&(mail=*)(!(telephoneNumber=*))) will match all entries that have an email address but do not have any telephone number.

**The Compare Operation**

The LDAP compare operation is the second of the interrogation operations. It allows a client to check whether a particular entry contains a particular attribute value or not. For example, if a client sends a compare request with providing DN, the server will return an affirmative response if the entry named by the DN contains the given value in the given attribute type; otherwise, it will return a negative response to the client.

**2.2.3.2 The LDAP Update Operations**

The LDAP update operations are used to add a new entry, to delete an entry, to rename or to modify an entry. There are four LDAP update operations, namely add, delete, rename, and modify. These four operations are useful in handling the directory.

The add operation allows the creation of a new entry in the directory. There are two parameters which are distinguished name (DN) and a number of attribute types and attribute values. In order to complete an add operation successfully, four conditions must be fulfilled:

- The new entry to be created in the directory must have a parent entry.
- The new entry name should be unique.
- The new entry must conform to the schema in effect.
- Access control must be permitted.

If these four conditions are fulfilled, a new entry can be created in the directory.
Structure of an LDAP Directory

The delete operation is used to remove an existing entry from the directory. It contains only one parameter which is the distinguished name (DN) of the existing entry. In order to remove an existing entry successfully, three conditions must be fulfilled [1]:

- The entry must exist in the directory.
- It should not have any children.
- Permission is required from access control.

If these three conditions are fulfilled, an existing entry can be removed without any problem.

The rename or modify operation allows the renaming or modification of an existing entry in the directory. This operation contains four parameters which are the DN of the entry, the RDN for the entry, an optional argument and the delete-old-RDN flag. In order to succeed in a rename or modify operation, three conditions should be fulfilled:

- The entry must exist in the directory.
- The new name must be unique.
- Permission is required from access control.

The delete-old-RDN flag, a Boolean flag, defines whether the old RDN of the entry is to be retained as an attribute of the entry. Different combinations of renaming and moving entries are represented by the following figs.

Fig. 2.2.2.3.2 (a). Renaming an entry without moving it.

Fig. 2.2.2.3.2 (b). Moving an entry without changing its RDN.
2.2.3.3 The LDAP Authentication and Control Operations

The bind operation allows a client to identify himself to the directory by providing an identity and authentication credentials; the unbind operation allows the client to terminate a session; and the abandon operation allows a client to indicate that it is no longer interested in the results of an operation it had previously submitted. LDAP provides two authentication operations, namely bind and unbind, and one control operation, abandon. The bind operation allows a user to identify him by providing a distinguished name and a number of credentials. Several kinds of bind methods are available. The simple bind method allows a client to identify himself by providing a DN and a password in clear text to the LDAP server. The server checks whether the password matches with the password value stored in the
Structure of an LDAP Directory

userpassword attribute type or not. If the password matches with that attribute value, the server will return a success code to the client. LDAPv3 also supports a SASL bind operation that is an extensible, protocol-independent framework for performing authentication and negotiation of security parameters. SASL allows a client to specify the type of authentication protocol that it wants to use.

The second authentication operation is the unbind operation. The unbind operation has no parameters. When a client issues an unbind operation, the server discards any authentication information it has associated with the client's connection, terminates any outstanding LDAP operations, and disconnects from the client, thus closing the TCP connection. The abandon operation has a single parameter: the message ID of the LDAP operation to abandon. The client issues an abandon operation when it is no longer interested in obtaining the results of a previously initiated operation. Upon receiving an abandon request, the server terminates processing of the operation that corresponds to the message ID. The abandon request, typically used by GUI clients, is sent when the user cancels a long-running search request. Note that it is possible for the abandon request (coming from the client) and the results of the abandoned operation (going to the client) to pass each other in flight. The client needs to be prepared to receive (and discard) results from operations it has abandoned but that server sent anyway. If we are using an LDAP SDK, however, we do not need to worry about this; the SDK takes care of this.

2.2.4 The LDAP Security and Access Control Model

The directory data should be protected against unauthorized access. The LDAP security model provides a set of protocol operations that allow users to authenticate so that they can search and update the directory data. As LDAP is a connection-oriented protocol, a secure connection is crucial. This model provides a secure connection between client and server. Before making a connection to the server, the client has to show that it has a valid identity in the directory. If a client sends a connection request to the server, the server will check whether it has some entity in the directory or not. If it has valid identification, the server will allow an LDAP connection; hence, it will get privileges from the server to access the directory, read data and update data if permission is granted. As an example, an LDAP client submits a distinguished name (DN) and a password to the LDAP server. The server locates the entry in the directory on the basis of its corresponding distinguished name and checks the password submitted by the client as to whether it is matched with the value stored in the userpassword attribute of the entry. If the check confirms the match, the client is authenticated. Otherwise, the authentication operation fails and the server sends an error code to the client.

2.2.4.1 Security

The LDAP uses SSL (Secure Sockets Layer) and TLS (Transport Layer Security) for security purposes to encrypt the entire data passing between a client and a server. SSL is successful technology for the World Wide Web. It secures electronic commerce and other transactions that depend on the transmission of data being hidden from eavesdroppers. On the other hand, TLS is comparable to Internet standard. LDAP uses TLS in order to provide a standard way for clients to begin encrypting all data flowing to, and from, the LDAP on the connection. As
an example, two companies which have a trading partner relationship can allow directory queries from their trading partners to travel over the Internet. If these two companies use TLS, it encrypts these queries and the results. So, each company can be assured that the directory data is protected while in transit over the Internet. Fig. 2.2.2.4.1 describes this scenario.

![Diagram of TLS secure transmission]

Fig. 2.2.2.4.1. TLS allows secure transmission of directory data over the Internet.

2.2.4.2 Authentication

LDAP provides several authentication methods. The best methods should be chosen which are suitable for the desired directory and unsuitable methods should be avoided. Possible methods are described below:

**Anonymous authentication:** in certain cases, we need to make directory data available to the users of the directory. In these circumstances, the users do not need to be authenticated. These users are called “anonymous” users. We can give permission of the directory to provide autonomous users by setting access control on information in our directory.

**Simple password:** the simple password method is used in such environments where data being protected is not a major concern. In this method, a client authenticates to the server by submitting a simple password. The password protection is also not a major concern. This method is appropriate for low-security environments.

**Simple passwords over SSL:** this method works well in environments where data protection is big concern and, therefore, requires high security. In this method, SSL is used to protect the connection over which a simple password is transmitted. For this method, we have to collect certificates for the servers, but not for the clients, and renew them when date is past. Both clients and servers need to implement SSL. This method has some restrictions. Firstly, not all clients and servers support SSL. Therefore, if we design a directory using SSL, it will not be widely usable by all types of clients. Sometimes, these inaccessible clients may be important for a user. Moreover, we have to implement SSL for both clients and servers. So, extra code should be included in the software; hence, software costs increase. The session also requires extra computing power and encryption demands extra cycles.
Certificate authentication over SSL: this method involves using the full power of SSL. This power enables us to protect the privacy of data, ensure the integrity of data and authenticate clients. The figure depicts the components of an SSL-based system supporting certificate-based client authentication.

The clients generate private and public key pairs, and issue certificates. These certificates and keys are managed and protected throughout their whole life cycle. A public key infrastructure (PKI) is needed to manage this life cycle.

Scheme via SASL: SASL is a generic mechanism for hooking in just about any authentication and security mechanism. This mechanism is best suited in the directory where Kerberos authentication database is used as leverage. When writing, it is only the Netscape Directory Server that supports a SASL interface that allows the writing of own plug-in module. However, the Netscape server does not support security-layer plug-ins: it supports only authentication plug-ins. To write an own SASL plug-in, each client must have a similar plug-in that understands the security mechanism.
2.2.4.3 Access Control

After applying one or more authentication schemes to identify the LDAP clients, it is important to decide how to use them to protect information contained in the directory. The access control provides this protection. Access control allows certain identities to access information in which all other identities are disallowed. This model is very useful to control users, groups and others for the administrators.

In general, access control is managed and specified by one or more access control lists (ACLs). Each ACL contains the following features.

One or more resources: resources, also called “objects”, which are used to control access. Generally, they are entries, attributes or values. As an example, cn specifies all the people entries in the directory.

One or more clients: clients are also called “subjects”. They are specified as authenticated directory users. A client access in the directory can be granted or denied by the administrators.

One or more access rights: access control rights enable administrators to manage all entries properly. These access rights determine what a client can do in the directory. For example, if a client is permitted “read and search” rights, the client will be able to read and search attributes from the entry.

Access control item (ACI) specifies an ACL, which can be placed anywhere in the directory tree. For example, an ACI attribute is shown below:

aci: (target ="ldap://dc=arius, dc=com") (targetattr="cn||sn||givenname||middleinitial||name") (version 3.0;acl "Anonymous read-search access"; allow (read, search, compare) (userdn = "ldap:///anyone");) aci: (target ="ldap://dc=arius, dc=com") (targetattr="cn||sn||givenname||middleinitial||name") (version 3.0;acl "Admin write access"; allow (write) (userdn = "ldap://cn=directory manager");)

Here, ACI is placed in a directory subtree rooted at the entry dc=arius, dc=com. It goes to that entry and all entries beneath it. The first value of aci allows read, search, and compare access rights to everyone accessing the directory, without any authentication. In the second value of ACI, the directory manager is granted write privileges to the directory.

2.2.5 LDAP APIs

LDAP API is important for designing a directory. It helps quick design, access and updating of the directory. Firstly, the University of Michigan has released a C programming library and several sample client programs built on this library. This C API was only available in API/SDK. However, the number of SDKs is growing, and additional SDKs are becoming available.

The LDAP C API for LDAP version 2 is documented in RFC 1823. The LDAP C API provides a number of core functions. Some of these are described below: ldap_search() is used to search for directory entries. ldap_compare() is used to see whether an entry contains a given attribute value. ldap_bind() authenticates to a directory server. ldap_unbind() is used to terminate an LDAP session. ldap_modify() makes changes to an existing directory entry. The
Structure of an LDAP Directory

ldap_add() is used to add a new directory entry. ldap_delete() deletes an existing directory entry. The ldap_rename() is used to rename an existing directory entry. ldap_result() retrieves the results of one of the previous operations. Moreover, JavaSoft [1] has developed the proprietary Java Naming and Directory Interface (JNDI). This API/SDK is used to define a common interface for accessing a number of different directory systems from a Java application or applet.
3 Design of an LDAP directory

Here, we will represent a comprehensive approach to a directory design and deployment. For a directory design, we should consider following matters:

- Determine what types of data we need
- Design Namespace
- Design Schema
- Design the Topology of Directory
- Design Replication Scheme

3.1 Data Design

Among all others things, a directory service is a specialized database. The kind of information that will store in a directory service is critical to obtain and manage that information. Planning and maintaining data will be the most complex and challenging aspects of deploying a directory service. Here, some techniques for identifying what data should be stored in a directory will be discussed and then, some potential sources for data and tips for maintaining good relationships with the other data sources that exist within our organization will be explored.

![Fig. 3.1. Data Sources, data elements and data values](image)

Data is a general term for the entire collection of information stored in the directory service. Sometimes, it invokes strong emotional and political responses because people care about their personal information, departmental information and organizational information. Moreover, those who write or deploy directory enabled applications have their own data related needs.

The directory service that will be designed is also the data source. The figure 3.1 shows the relationship between data sources, data elements and data values.
3.1.1 Data Elements Characteristics

Data element characteristics are given below:

- Format
- Size of data values
- Number of occurrences
- Data ownership
- Consumers
- Dynamic Versus static
- Shared versus application specific
- Relationship with other data elements

From the following points of view consider each data element planned to be included in a directory service. To save time during the schema and namespace design stages and avoid deployment problems.

**Format:** the natural format of the information data elements can be grouped. For example, a person’s name is always a textual data, but a telephone number consists of digits.

**Size of Data values:** size defines to the number of characters or bytes that a data element value contains. If the approximate size of each data value is known, it will help to design a directory with more entries.

**Number of Occurrences:** for each data element, one should know how many data values will be stored. For example, a person will usually have only one user ID but may have several phone numbers. These types of information will be helpful in planning the directory’s capacity. It is also useful for replicating and synchronizing the data.

**Data Ownership:** the data ownership issue is one of the most challenging aspects of a directory design. For the access control, the privacy and the security, the data ownership is important, because it is important to know accurately who should be allowed to view or modify the data.

**Consumers:** the people who use the directory-enabled applications are referred to consumers. When it is planned for a directory replication, topology, and managing the relationship between a directory service and others data sources, it will be helpful to know the consumers of each data element.

**Dynamic versus static data elements:** which data elements are dynamic and which are static. For a directory design as well as for a capacity design, we need this information.

**Shared versus static data elements:** many applications use some common data elements. If a data element is used by only one application and the data values are large or accessed frequently, one should consider keeping the values outside our directory service to avoid the performance problems when accessing the values.
Relationships with other data elements: it is important to know which data elements relate to the same kind of object, when we design schema and namespace for directory service.

3.1.2 Data Sources

In a directory, the data may come from various sources. If there is a list of data elements to be included in a directory service, data sources will help to know from where the values for data elements will be obtained and how will be managed the relationships with other data sources.

Fig. 3.1.2. Various sources of directory data.

Common data sources are given below.

- Other directory services
- Network operating systems running within local area networks
- Online-based databases
- Files and spreadsheets that hold data in an electronic form
- Applications
- Administrators
- End users

For every data element we planed to include in a directory, one should decide the following:

- What data source should be considered in order to obtain the initial data values?
- Who will get access the data element?
- Who will get permission to modify the data element?
- Would the data element be updated from other data sources on an ongoing basis?
3.1.3 Managing Good Relationships with other Data Sources

For maintaining good relationships with other data sources, it is important to have some idea as to how the data held in a directory service relates to that stored in other data sources within an organization. Many organizations suffer from data redundancy problems, where data elements are stored in multiple and uncoordinated directory systems.

3.1.4 Synchronization

Synchronization [1] is the process in which a data protection manager (DPM) transfers data changes from a protected file server to a DPM server. Synchronization differs from a replication in the protocols, schema and data formats. Different vendors have different synchronization tools. For example, the Netscape Directory server package includes a Windows NT synchronization service that provides two-way synchronization between “Windows NT Domain Controllers” and the “Netscape Directory Server”.

3.1.5 Batch Updates

Batch updates are a kind of “loosely coupled” synchronization. Batch updates occur less often, and they involve the merging of data that comes from different sources.

3.2 Schema Design

The schema is a set of rules which determine what types of data that can be stored in a directory or a database. It is very important because it helps maintain the integrity and quality of the data stored in a directory service. It also helps to reduce the duplication of data stored in a directory. A directory server checks the contents of an entry against the schema rules if it stores a modified or new entry in the directory.

One of the main purposes for a schema is to guarantee that the redundant data can not be stored in the directory. For example, if a person’s name is stored in a different directory attributes, this would result in wasted storage space and values that should be the same but are different. Therefore, it would ultimately lead to a lot of confusion on the part of applications and end users.

Moreover, schemas are used to impose constraints on the size, range, and format of data values stored in the directory. For instance, according to the Internet mail standards, email address values should ensure to a specific format (addr@domain) and should use a restricted set of characters. In many cases, schema rules compel simple restrictions, namely "this value must be a character".

3.2.1 DIT Layout

The DIT (Directory Information Tree) layout is described below [4]:

3.2.1.1 Top Levels

There are two common forms used for the higher levels in the tree, namely geographic and domain-based. The geographic form was used in the original X.500 standards, as it is most suitable for global-scale “white pages” searching. Domain-based trees were introduced to provide a simple way to map Internet domain names into the DIT. However, for internal
Design of an LDAP Directory

directory services, the Domain Component tree is the best choice. The two forms can co-exist under the same virtual root if required. The figure describes these two forms.

![Diagram showing Geographic and Domain Component naming](image)

Fig. 3.2.1.1 Geographic and Domain-component naming

### 3.2.1.2 Lower Levels

The lower levels of DIT are described below:

**Where to put the people**

All categories of people entries of an organization should be placed in a specific part of a directory. For example, it seems like a good way if all the people in the Customer Service Department are listed under that department's own entry, hence, it should be easy to hand over control to the departmental administrator.

**Representing the organization**

The structure of the organisation can be represented using organizationalUnit and organizationalRole nodes. Groups

People form themselves into groups in order to meet some common purposes. The directory might have to represent committees, mailing lists, clubs, people with particular powers or privileges, etc. Normally, groups are represented by LDAP objects with the groupOfNames class. In these objects, member attributes represents group members. These attributes must have multiple values. The groupOfNames class makes the member attribute mandatory. Therefore, it is not possible to create an empty group whose values are the DNs of themembers themselves. On the basis of an organisation, this might be a single dc=groups node, or it might be a subtree divided into different categories.

**Application-specific data**

Most software applications need to store configuration data somewhere. For example, for network applications, there are advantages to storing configuration data in the LDAP directory. Multiple servers can easily access the configuration. Also, it may be possible to make the application servers effectively “dataless” so they can be replicated easily and do not need to be backed up.
3.2.2  Elements of LDAP Schemas

In a LDAP directory, a schema contains attribute types, attribute syntaxes and object classes which are described below.

3.2.2.1  Attributes

Directories entries contain a number of attribute types and values. Attribute type contains specific data elements, particularly names, phone numbers, mail and so on. An attribute type specifies the following [1]:

- The name of attribute type must be unique
- It must be uniquely identified by an object identifier (OID)
- Whether the attribute is multivalued or single-valued should be mentioned
  An attribute contains a set of syntax and matching rules
- Limitation on the size of the values.

An attribute value represents the assigned value. For example, the attribute “cn=Nayan” means that the attribute type is cn (common name) and the attribute value is Nayan.

Different kinds of attributes are described below:

3.2.2.1.1  Naming attributes

Most of the naming attributes are derived from the generic name attribute type. The length limit of these attributes should be 32768 characters, the UTF-8 character set, and a set of case-insensitive matching operations.

**cn (Common Name)**

The Common Name (cn) attribute contains a name of an object. If it represents a person, it is typically the person’s full name. These attributes can be single valued or multi-valued. For example, cn: Brian Stanley Walker is multi-valued and cn: Walker is single valued. cn is mandatory in person entries, and is one of the most important attributes as it is commonly specified in white pages searches.

**sn (Surname)**

The attribute sn contains the family name of a person. sn is mandatory for person entries. It can hold multiple values.

**givenName**

In order to contain the part of a person’s name the givenName attribute is used. It is not their surname or middle name. It is hardly used.

**personalTitle**

This attribute represents a personal title for a person, namely “Mr”, “Mrs” etc. It comes from RFC1274 and has a length limit of 256 characters.
Design of an LDAP Directory

**o (Organisation Name)**
This attribute specifies the name of an organization. It can be multi-valued. In case of LDAP, it gets a limit of 32768 characters.

### 3.2.2.1.2 Descriptive attributes
Different kinds of descriptive attributes are described below:

**jpegPhoto**
This attribute is used to store one or more images of a person using the JPEG File Interchange Format [JFIF]. The size limit is 250000 bytes.

**Description**
A human-readable description of the object is contained by this attribute. Its size limit is 1024 characters.

### 3.2.2.1.3 Postal attributes
Different kinds of postal attributes are described below:

**postalAddress**
This attribute specifies a person address where he or she resides. Its size limit is 6 lines of 30 characters each.

**postalCode**
This attribute represents the postal code of the named object. It is a part of the objects’ postal address. Length limit is 40 characters.

**c (Country Name)**
This attribute specifies a country name. It holds a two-letter ISO 3166 country code. This attribute is single valued.

### 3.2.2.1.4 Telecommunications attributes
Different kinds of telecommunication attributes are described below:

**mail (RFC 822 mail address)**
It is the normal electronic-mail address. It must be given in full RFC822 format (e.g. engg.nayan@yahoo.com). Size limit is 256 characters.

**telephoneNumber**
This attribute is used to represent telephone numbers. This allows things like: +46 765834203 where all characters must be integer. Size limit is 32 characters.

### 3.2.2.1.5 Authentication attributes
Different kinds of authentication attributes are described below:
Design of an LDAP Directory

**uid (User Identifier)**
The Userid attribute type represents a computer system login name. Length limit is 256 characters.

**userPassword**
This attribute is used to authenticate a person in the directory services. It can hold multiple values, each of which is limited to 128 characters.

### 3.2.2.2 Attribute Syntaxes and Matching Rules
Attribute syntaxes specify exactly how data values are represented and how comparisons between values are made. Matching rules are used to make comparisons between attribute values.

### 3.2.2.3 Object Classes
The same kinds of information are grouped together. Object classes are used for this purpose. Generally, an object class gives the model of real-world objects, specifically, a person, a printer, a network device etc. For example, a person object class may contain many attribute types that are related to people’s personal information, namely a common name (cn), a surname (sn), a phone number, a mail address and so on. In an LDAP, an object class specifies the following.

- An object class name should be unique.
- It should have a set of mandatory attribute types.
- It should have a set of allowed attribute types.
- It should be a specific type (structural, auxiliary, or abstract).

There are three types of object classes, namely structural classes, auxiliary classes and abstract classes. Structural classes specify the fundamental aspects of an object. Auxiliary classes are used to append a number of related attributes to an entry that already belongs to a structural. On the other hand, abstract classes are hardly used, and only for those classes that need to support an LDAP's basic information model.

### 3.2.3 Comparisons between Data Design and Schema Design
Data refers to the entire collection of information stored in a directory. It makes a directory service more interesting. If a directory contains varieties of data about an application, it becomes more useful and effective. The term data element refers to the pieces of data. Data sources refer to any system that stores a collection of data elements. In an application, there may be many sources of data. Furthermore, the issues surrounding data sometimes bring to mind strong emotional and political responses since people care about their personal information, their department's information, and the organization's information. Additionally, those who design or deploy directory-enabled applications have their own data-related requirements. Some of the different requirements will inexorably conflict. So, proper ways of
Design of an LDAP Directory

selecting data sources are important to design a directory effectively. Data design handles these ways so that the application can meet the data needs for all.

After collecting data from various sources of an application, it is important to keep these data in a directory. Some basic rules are needed for storing these data in the directory. The schema design provides such rules for processing and storing data in the directory. It uses attributes, object classes and entries to represent data. Data elements are stored as attribute types in the directory. In LDAP terms, data values are called attribute values. An object class contains some common attributes. Some attributes and object classes together represent an entry while a directory is a collection of entries.

3.2.4 Schema Checking Process

If one wants to add a new entry to the directory or modify the existing entry, the directory server that processes the request goes through a schema checking process before committing the add or the modify request. This process is done before the directory database itself is altered and ensures that all new or modified directory entries conform to the schema rules.

When the schema process is done, the following steps are performed by the server:

- It checks all new or modified values in the syntax rules.
- It checks at least one value for the object class attribute that is present.
- It ensures at least one value for each of the mandatory attributes that is present for every object class.
- It verifies every attribute to make sure that it is allowed by one of the object classes.

3.2.5 Schema Maintenance and Evolution

According to new applications, we need to change the schema over time. When a directory service becomes popular and new applications are rapidly being proposed, it may be difficult to keep up the previous design with the demand for new schemas. A more decentralized approach to schema design would be needed.

3.3 Namespace Design

The namespace design is one of the most important tasks for designing a directory service. The namespace provides the basic means by which information is referenced in a directory. A well defined namespace can lead to:

- An easy way of maintaining data.
- Flexible setting of access control and replication policies.
- Namespace that can satisfy a large range of directory-enabled applications.
- A natural way of navigation through the directory.
- Flexibility for directory users and administrators.
Design of an LDAP Directory

Moreover, a poorly designed namespace can lead to administrative hassles when the directory entries change names, replications or access control requirements change or a user tries to find information.

3.3.1 The Structure of a Namespace

An LDAP model defines a flexible namespace framework. The LDAP namespace model is inherited from the X.500 directory standard, which is intended to be used in a fairly rigid hierarchical directory service. Typically, the X.500 namespace starts with countries at the top of the namespace, perhaps followed by states in the United States, and then the organization.

![Diagram](image.png)

Fig. 3.3.1 (a). A typical X.500 namespace

The LDAP model is hierarchical or tree-structured. The simplest hierarchical case of a one level flat namespace is also allowed. The LDAP does not support to a graph structure in which an entry might be both child and parent of the same entry.

The namespace can be designed as flat or hierarchical.

Flat Namespace

Name changes are cumbersome and burdensome tasks for administrators. The flat namespace is used in such cases where names are changed less frequently. One should design a namespace as flat as possible, because flatter names change less and are easier to administer.
Hierarchical Namespace

In order to enable data partitioning among multiple servers, replication, and certain kinds of access control, hierarchy can be used. Another reason to introduce hierarchy is to enable the distribution of administrative authority via access control. For instance, suppose one wants to permit an administrator from the marketing department to have control over marketing entries, the engineering administrator to have control over engineering entries, and so on. However, many directory products allow this kind of administrative distribution of control only at hierarchical namespace points. In these systems, various access control rules cannot easily be applied to the same subtree.

3.3.2 The Purposes of a Namespace

Namespace defines the means by which directory data is named and referenced. Here, we will try to summarize the more common purposes of a namespace. These are described below:
Design of an LDAP Directory

**Data Reference:** this provides the means by which directory data is referenced. It is important for two reasons. One is that there must be a way for directory clients to refer unambiguously to a directory entry when retrieving or modifying the information associated with it. The other is that the directory namespace provides a compact and efficient way to support a group of directory entries that refer to one another.

**Data organization:** the namespace defines the way of organizing data. For example, it would be wise if we placed all entries relating to people in one portion of the namespace, and entries corresponding to device (such as printers) in another, and groups in another part of the namespace. Again, the subdivision of an organization is also possible, perhaps based on geographical or organizational information.

![Diagram of namespace organization](image)

**Fig. 3.3.2 (a).** A namespace example for data organization

**Data partitioning:** the namespace can partition the data directory in one server or it can divide into multiple servers. Mostly, partitioning is possible only in a certain well defined namespace. If a namespace does not hold partitioning points, we can not partition the data.

![Diagram of namespace partitioning](image)

**Fig. 3.3.2 (b).** A namespace example for partitions, replication and delegation through access control.
Access control: access control can be affected by the namespace. Some of the products allow the setting of the access control only at namespace branch points. This is true when delegating authority to some of the data. It is possible only if the data is divided on a sub tree basis.

Application support: the purposes of a namespace have all been related to the operation of the directory itself. Finally, it will be mentioned that the namespace should support the application that design a directory for in the first place.

A namespace is required to reference entries and provides features, namely groups, and it should support the applications using the directory.

3.4 Topology Design

An LDAP directory service, it can support a distributed directory where the complete directory tree is spread across multiple physical directory servers. A directory server topology describes if divides the directory tree among physical servers and how allocate those servers among an organization’s physical locations. Good choices of a directory topology will help in the following ways [1].

- It will be possible to achieve the best performance for the directory enabled applications.
- It will enhance the directory availability.
- It will be possible to manage a directory better.

![Fig. 3.4. A distributed directory](image)

3.4.1 Designing of Directory Server Topology

Optimal directory sever topology depends on following four factors:

- Directory enabled applications use, or planed use and the expectations of users of those applications.
- Directory severs software and capabilities.
- Physical network topology.
Design of an LDAP Directory

- Directory namespace.

Directory enabled applications: where the directory network traffic will be coming from. To get this information, one should start by taking an inventory of the directory applications. Some common types of directory enabled applications are given below:

Authentication and login applications: these types of applications verify the user ID and password.

Authorization applications: these types of applications make some sort of authorization checks for the purpose of accepting or denying access to some resources.

Address book applications: these types of applications are usually embedded inside electronic mail applications.

Messaging applications: these types of applications use the directory to make a route electronic mail message to their destinations.

Directory server software and its capabilities: the directory servers have some practical limitations in the following number of entries that can reasonably be stored in a partition:

- Number of partitions that can be held on a server.
- Number of entries that can be held on a server.
- Number of subordinate references that can be maintained at a given level of the DIT.
- Number of replicas of a partition that can be maintained.
- Number of indexed searches that can be serviced per second.
- Number of concurrent client connections that can be supported.

Creating a Map of Physical Network: one should make a map of the organization physical network. Then, he should focus on the components for the connection of an LAN. Each LAN segment with its physical location should be labeled and connected between LANs with the bandwidth, latency and reliability of the connection.

### 3.5 Replication

Replication is the process of keeping a directory data in different servers. If the data is kept only on a server, it will be risky. The data will be lost when the dedicated server fails. So, keeping data in different servers is a good policy. It also increases the reliability and the performance of a system. In a replication, when any single server fails, another replicated server handles the directory clients and the directory-enabled application programs. Therefore, the directory clients and directory-enabled application programs could contact a different server for their directory services. Hence, the directory data becomes safe and existing services remain intact. Figure 3.5 (a) depicts how a client can use this redundancy to obtain a service from a replica at the time of a server failure.
Moreover, when the directory content is replicated, the performance of the directory also increases. Then, the directory will be able to handle more directory clients by distributing the load across many servers. Figure 3.5 (b) illustrates such scenario.
3.5.1 Replication Strategy

The replication strategy describes the flow of the data from a server to another server. It also describes the way servers interact at the time of generating updates. There are three main approaches of replication, namely single-master replication, floating-master replication, and multi-master replication.

Fig. 3.5 (b). Distributing directory client load.
3.5.1.1 Single-master Replication

There is only a server in single-master replication that is responsible for holding a writable copy of a directory entry. This server is the master server which can perform write operations. Read-only copies of that entry are contained by all other replicas. The other servers may perform a search, compare or bind operation. The figure 3.5.1.1 shows the strategy of a single-master replication.

3.5.1.2 Floating-master Replication

The floating-master replication is used to avoid a single point of failure. In a floating-master replication, there is a writable copy of a directory entry for a given time. When the read-write server is not available, a new read-write server is chosen by a voting algorithm in which the remaining servers collectively agree on a server to become the new master. The figure 3.5.1.2 shows a floating-master replication.
3.5.1.3 Multi-master replication

Like a floating-master replication, a multi-master replication avoids a single point of failure. It provides more reliability. It allows more than one server to accept read-write operations. In this replication, more than one read-write copy is always available. So, clients may submit a write operation to any master replica. Then, it becomes responsible to ensure other cooperating servers that changes have been updated to all servers. The figure 3.5.1.3 describes how two replicated servers are capable of handling a client’s write requests.
Fig. 3.5.1.3. Multi-master replication.
4 A Library System- Problem and Solution

LDAP directory servers are usually centralized hierarchical databases of indexed information about a given domain. The information can for example be useful for network administration, especially for databases containing information about network users and network devices. In big organizations in which there are a great number of users and a network is divided into several sub-domains, the performance of this kind of solution may suffer from degradation in moments of high pick usage, e.g. beginning of a work day. A straightforward way to solve this problem is to run the service on more powerful computers. However, this is not a scalable solution, as the size and the dimension of LDAP directories may grow exponentially if additional concepts are added to it. So, a more smart solution has to be used. The purpose of this thesis is to propose the architecture of a distributed LDAP directory, in which a central LDAP server just maintains pointers to other LDAP servers, which provide information about the libraries, with different kinds of information, specifically information of users, various existing services, librarians, administrators, and different kinds of books. The different information is saved in different LDAP servers, which are coordinated by a central one that just distributes the searches or detailed information about each domain. Hence, the thesis will discuss the possible alternatives to implementing such distributed and correlational LDAP directories and select one as proof of concept to be implemented by the group.

4.1 Distributed LDAP Directory based Library Systems

The use of directories is increasing day-by-day in many ways. So, a developed and well designed directory can ease our daily lives in many ways. An LDAP directory is a well-functioned and easy to access a directory model in which the LDAP protocol provides many functions to handle and access the directory. In our thesis, we will design a distributed LDAP, directory-based, library system. In this system, there will be a central library that will control and manage other subordinate libraries. Each library will contain an LDAP directory and the central LDAP directory in the central library will control the data of other directories. As an example, a client of a local library wants to have a specific book. After searching, if he finds that the desired book is not available in the directory of that library, he will ask the administrator to obtain this book. Then, the administrator will contact with the administrator of the central library. At first, the administrator of the central library will check his own directory to see whether the required book is available or not. If the book is not available in the central library, the administrator will search for other subordinate local libraries, because he will have access rights of directories in all local libraries. In this way, data will be distributed and centrally controlled in our proposed system. In our thesis, we will focus on the following research questions:

1. In which way will the directories coordinate and share data between each other?
2. How reliable will the system be?
3. How efficient will the system be?
4.2 Requirements on Distributed LDAP Directory based Library System

At first, we will design LDAP directories for different libraries. In order to design a directory, we have to meet the following requirements:

User needs: in our directory, the user will be able to search and order books. We will provide some basic information about the specific book so that the user can easily get some ideas about the book. In the directory, a user name, some information of a user, namely an e-mail address, phone number, mailing address and password, will be stored. The user will get access to the belonging directory by providing his user name and password. It will get only read access to the book entries. It will not be able to edit or write anything.

Application needs: in our application, we will put different categories of books in every directory. In a book entry, we will provide the book name, the author name, the book edition, the book identifier number, published place, the book description and the “See Also” attribute, so that the users can get a brief idea about the books.

Administration needs: the system administrators of the relevant directory will be able to create, edit, delete and modify the entries. They will manage the directory and control the user access. They will get all rights reserved in the directory. The librarian will be able to search and update the information of entries, but they will not be able to create a new entry or delete an existing entry.

4.3 Architecture of the Distributed LDAP Directory based Library System

The architecture of a distributed LDAP directory based library system is depicted in figure 5.1 (a). In this system, there is a central library named “Dhaka Central Library” and three other subordinate libraries, namely “Chittagong Library”, “Khulna Library” and “Rajshahi Library”. The central library will control all subordinate libraries.
Fig. 5.1 (a). Architecture of distributed LDAP directory based library system.
In each library, there will be an administrator, one or more librarians and many users. Additionally, each library will contain various books and provide many services to the users. All this information will be stored in the directory. The user will be able to communicate with the administrator of his own library. However, a user of a specific library will not be able to communicate with the administrator of the other libraries. The administrator of the central library will be able to communicate directly with the administrators of its subordinate libraries. However, the administrator of a subordinate library will not be able to communicate with the administrator of other subordinate libraries. For example, if an administrator of a subordinate library wants some information, he will, at first, communicate with the administrator of the central library. The central administrator will give feedback to that administrator, collecting information from other libraries if it does not have such information. The figure 5.1 (b) describes how a user and an administrator will communicate with each other. We have used a client/server protocol in order to implement this communication. A client requests the services and a server provides services to the clients. In our system, the clients are considered as users and the administrator as a server. In order to implement this client/server protocol, we have used the socket programming in Java programming language. Our socket programming uses the TCP/IP protocol.

Fig.5.1 (b). Users and administrator communicate as client/server

In the computer networking, a network socket [2] is an endpoint of a bidirectional inter-process communication flow across a computer network.

A socket address [2] is comprised of an IP address (the location of the computer) and a port (which is mapped to the application program process) into a single identity. A local socket address contains a local IP address and a port number. The remote client communicates
with the server by providing its corresponding IP address. The figure 5.1 (c) describes how a client/server protocol works in a TCP socket.

Fig. 5.1 (c). TCP Socket flow diagram
Sockets are usually implemented by an API library, namely Berkeley sockets. Some examples of functions or methods typically provided by the API library are given below [2]:

- **socket()** creates a new socket of a certain socket type by providing an integer number.
- **bind()** is used on the server side. It associates a socket with a specified local port number and IP address.
- **listen()** is used on the server side. It causes a bound TCP socket to enter listening state.
- **connect()** is used on the client side. It causes an attempt to establish a new TCP connection.
- **accept()** is used on the server side. It accepts a received incoming attempt to create a new TCP connection from the remote client. So, a new socket associated with the socket address pair of this connection is created.
- **send()** and **receive()** are used for sending and receiving data to/from a remote socket.

In this system, the users will search for books and existing services from the belonging directory. The librarian will update directory data and the administrator will control and manage the directory data and the user access. Only the central administrator will have full access to other directories; hence, he will be able to update and search for the data of other directories.

### 4.4 The LDAP Directory Design of our System

In order to design an LDAP directory, we have used “Apache Directory Studio”. This is a complete directory tooling platform intended to be used with any LDAP server [10]. However, it is particularly designed for the use with the “Apache Directory Server”. It is an Eclipse-based application that is composed of several Eclipse (OSGi) plugins. These plugins can be easily upgraded with additional ones. These plugins can even run within Eclipse itself.

**Data design**: in each directory, we have designed data for five different types of entries. We have put an administrator name and other personal information in the Admin entry. In the librarian entry, we have kept various personal information of the librarian. Moreover, in a user entry, we have assigned the user name, password and some other information in order to identify him. However, the books and services entries provide us the biggest sources of data. The book entries contain different book names, writer names, editions and some other information so that any user can get a brief idea about the books. The services’ entries provide much information about the existing services in the library. The figure 5.2 (a) shows the sources of data of a directory, such as “Dhaka Central Library”.
Fig. 5.2 (a). Directory data source diagram

Schema design: the schema used in each directory consists of two basic sets of schema definitions, one representing people and the other “Central Library Book Store”. The schema used for representing people has been taken from the standard person schema definition.

In our directory, we have created five types of entries. Firstly, we have created an entry for administrators. Secondly, entries have been created for book names. Thirdly, entries have been created for services that are provided for the users by a specific library. Fourthly, entries have been created for librarians. Finally, entries have been created for library users. Different types of entries have been given different kinds of attributes and access rights.

An administrator has been given all rights reserved in the directory. He will create a new entry. He will delete an existing entry. Moreover, he will update and manage the directory data. Also, he will control a user or a client access in the directory.

In the administrator entry, we have used five types of object classes, namely organizationalPerson (structural), initOrgPerson (structural), person (structural), accessControlSubentry (auxiliary) and userSecurityInformation (auxiliary). Also, various attribute types and values are declared. The administrator entry is shown below:

```
dn: cn=Nayan,uid=admin,ou=people,o=Dhaka Central Library
```
Here, in this entry, the distinguished name of the administrator is dn: 

cn=Nayan,uid=admin,ou=people,o=Dhaka Central Library. Moreover, there are seven different attributes. For example, attribute (cn: Nayan ) represents that the attribute type is cn (common name) and the attribute value is Nayan.

In the librarian subschema, we have put entries for librarians. As an example, an entry for librarian is shown below:

dn: cn=Javed,ou=Librarians,ou=people,o=Dhaka Central Library

Here, in this entry, the distinguished name of the administrator is dn: 

cn=Nayan,uid=admin,ou=people,o=Dhaka Central Library. Moreover, there are seven different attributes. For example, attribute (cn: Nayan ) represents that the attribute type is cn (common name) and the attribute value is Nayan.
cn: Javed
sn: Alam
mail: javed@yahoo.com
departmentNumber: et3345
homeTelephoneNumber: 843673473
homePostalAddress: Road no-4, house no-56
description: A librarian.
userPassword: Plain text password

Here, the distinguished name for this entry is dn: cn=Javed, ou=Librarians, ou=people, o=Dhaka Central Library. The above entry contains five object classes, namely top (abstract), accessControlSubentry (auxiliary), inetOrgPerson (structural), organizationalPerson (structural) and person (structural). It also contains eight different attributes. For example, attribute (sn: Alam) represents that the attribute type is sn (surname) and the attribute value is Alam.

In the subschema named users under people, various entries for users have been set. For instance, an entry for a user is described below:

dn: cn=Prosanto, ou=users, ou=people, o=Dhaka Central Library
prescriptiveACI: { identificationTag "", precedence 0, authenticationLevel none, itemOrUserFirst userFirst: { userClasses { allUsers }, userPermissions { { protectedItems { }, grantsAndDenials { grantBrowse, grantFilterMatch, grantRead } } } } }
objectClass: top
objectClass: accessControlSubentry
objectClass: person
objectClass: inetOrgPerson
objectClass: organizationalPerson
cn: Prosanto
sn: Kumar
mail: prosanto@yahoo.com
telephoneNumber: 004678453646
departmentNumber: IDE
description: A general user
userPassword: Plain text password

The distinguished name for this above entry is dn: cn=Prosanto, ou=users, ou=people, o=Dhaka Central Library. This entry contains five object classes and seven different attributes. As an
example, the attribute (mail: prosanto@yahoo.com) represents that the attribute type is mail whereas attribute value is prosanto@yahoo.com.

The subschema “ou=Central Library Book Store” contains two different subschemas named “Bookself” and “Services”. The Bookself subschema contains different kinds of books. For example, an entry for a book is described below:

dn: ou=documentTitle=Code Complete, Bookself, ou=Central Library Book Store, o= Dhaka Central Library

objectClass: document

objectClass: top

documentTitle: Code Complete

documentAuthor: Mr. Malin

documentVersion: 3\textsuperscript{rd} edition

documentPublisher: July 7th 2004 by Microsoft Press (first published 1993)
description: This is a good book for code completing.

seeAlso: This book is available at this moment in our library bookshelf.

In the above entry, the distinguished name is “dn: documentTitle=Code Complete, ou=Bookself, ou=Central Library Book Store, o= Dhaka Central Library”. It has two object classes, namely document (structural) and top (abstract). Moreover, it contains ten different attributes. For example, the attribute (documentTitle: Code Complete) means that the attribute type is documentTitle and the attribute value is “Code Complete”.

**Namespace Design**: while designing a directory, designing a directory namespace is one of the most important tasks [1]. The directory namespace provides the basic means by which information can be referenced in the directory. An adequately designed namespace leads to easier data maintenance, flexibility in access control, flexibility in natural navigation through the directory and so on. On the other hand, a poorly designed namespace causes hassle both for administrators and users. The figure 5.2 (b) depicts the namespace design of our directory where the data is shown hierarchically. In this namespace design, a DIT (Directory Information Tree) is the root of all entries. The schema named “o=Dhaka Central Library” is partitioned into two sub schemas called “ou=Central Library Book Store” and “ou=people”. The people subschema is again partitioned into three subschemas, namely “uid=Admin”, “ou=Librarian” and “ou=users”. Under the Admin, an entry is created for an administrator. A Librarian subschema contains entries for librarians while subschema named “ou=users” provides entries for library users. A subschema named “ou=Central Library Book Store” is partitioned into two subschemas called “ou=Bookself” containing different kinds of books, and “ou=Services” containing various existing services in the library.
Security and access control: the Apache Directory Server [12] uses SSL technology (Secure Socket Layer) or its standardized successor TLS (Transport Layer Security, RFC 2246) in order to secure transport layer. These technologies are useful for the protection of data against eavesdropping and modification, when on transit between a client and a server (data integrity), and the authentication of a server toward a client with the help of a certificate. There are two approaches to perform the transport layer security. One is ldaps (LDAP over SSL/TLS, port 636) and another is StartTLS (extended operation). The first approach is compatible to HTTPS and injects an SSL/TLS layer between the TCP/IP protocol and LDAP. In the second approach, firstly, a client establishes a normal LDAP connection. However, it switches to secure communication, with a special request (extended operation StartTLS). Both approaches need to configure the server with an appropriate certificate.
In our directory, different categories of people have been given different access rights. The administrator has been given all rights reserved in the directory. The librarian has been given data update rights. The users have been given read, search and match rights. We have declared an objectClass named accessControlSubentry in the entry that provides attributes for access control. These access controls are performed by ACI. The ACI declared for administrator entry is shown below:

prescriptiveACI: { identificationTag "", precedence 0, authenticationLevel none, itemOrUserFirst userFirst: { userClasses { allUsers, thisEntry, parentOfEntry }, userPermissions { { protectedItems { entry, allUserAttributeTypes, allUserAttributeTypesAndValues }, grantsAndDenials { grantInvoke, grantBrowse, grantFilterMatch, grantRename, grantCompare, grantReturnDN, grantImport, grantAdd, grantModify, grantRemove, grantDiscloseOnError, grantExport, grantRead } } } } }

The ACI declared for librarian entry is shown below:

prescriptiveACI: { identificationTag "", precedence 0, authenticationLevel none, itemOrUserFirst userFirst: { userClasses { allUsers }, userPermissions { { protectedItems { }, grantsAndDenials { grantInvoke, grantBrowse, grantFilterMatch, grantRename, grantCompare, grantReturnDN, grantImport, grantAdd, grantModify } } } } }

The ACI declared for user entry is shown below:

prescriptiveACI: { identificationTag "", precedence 0, authenticationLevel none, itemOrUserFirst userFirst: { userClasses { allUsers }, userPermissions { { protectedItems { }, grantsAndDenials { grantBrowse, grantFilterMatch, grantRead } } } } }
5 Experimental Results and Performance Analysis

5.1 LDAP Server

There are many commercial LDAP servers, including Netscape Directory Server, IBM Tivoli Directory Server and Novell LDAP Services. However, we have chosen the Apache Directory Server as it is an open source program. Apache Directory Server is an embeddable directory server entirely written in Java. It has been certified the LDAPv3 compatible by the Open Group. In addition, it supports Kerberos 5 and the “Change Password Protocol”. Moreover, it has been designed to introduce triggers, stored procedures, queues and views to the world of LDAP. It provides ease of data searching, editing and modifying. Also, it provides strong security protocols, namely SSL and TLS.

An LDAP server provides directory services to the clients. The LDAP clients send requests to the LDAP server and the server sends response to the clients.

5.2 LDAP Client

The clients send the service requests to the server. In our system, we have used the Xplorer software as client software. Using this software, the clients search directory data. The JXplorer [11] is an open source LDAP browser originally developed by “Computer Associates' eTrust Directory Development Lab”. It is a standard compliant general purpose ldap browser that can be used to read and search any LDAP directory, or any X500 directory with an ldap interface. It is available for immediate free download under a standard OSI-style open source license. It is a fully functional piece of software with advance security integration and support for more difficult and obscure parts of the LDAP protocol.

The overall client-server architecture used in our system used is shown in figure 6.2. A number of client machines are connected to a server machine.

![Diagram of LDAP client/server interaction](image)

Fig.6.2. LDAP client/server interaction.
Experimental Results and Performance Analysis

The server machine should have more RAM, a fast disk space, powerful processors, but does not need to have a good display. On the other hand, the client machine does not need to have more RAM, a fast disk space and powerful processors. Simple personal computers are adequate for client machines.

5.3 Experimental results

Common LDAP operations are modify, add, delete, compare and search. The search filter for the search operation was created from the interface address of interest, and the related policy object. A simple LDAP search requires four operations successively, namely ldap_open, ldap_bind, one or more ldap_search operations, and ldap_unbind. The figure 6.3 (a) describes that the ldap_open operation is used to open a connection to the directory server. It returns a session handle for future use. The ldap_bind operation is used in order to authenticate clients. By the bind operation, an LDAP client gets chance to identify himself to the directory server by using a “Distinguished Name” and some authentication credentials (a password or other information). In our experiments, we have used password authentication, although the LDAP supports a variety of authentication methods. After the successful completion of a bind operation, the directory server remembers the new identity until another bind is done or the LDAP session is terminated by calling the ldap_unbind. The server uses the identity to make decisions about what kind of changes can be made to the directory. The ldap_search operation commences a LDAP search, specifying the criteria that entries fits into the associated filter. Finally, the ldap_unbind operation terminates an LDAP session.

Fig.6.3 (a). Successive steps of LDAP search operation
Experimental Results and Performance Analysis

From the figure 6.3 (a), we can define some latency measures [3]. The time from the sending of ldap_open request until ldap_bind operation is successful, is called the connect time. The processing time is considered as the time required for the ldap_search operation and the data transfer time for the retrieved results to be returned to the clients. The response time is defined as the total time required for an LDAP search operation, from ldap_open to ldap_unbind. The performance of the LDAP service is reflected by the connecting, processing and response latencies, and the throughput of the server. The throughput is represented by the average number of requests served per second.

We have designed the LDAP directory using Eclipse based Apache Directory Studio. The screenshot of Apache Directory Server configuration is shown below.

Fig.6.3(c). Screenshot of Apache directory server configuration.

The “Max. Time Limit” is used to choose the maximum time that should last a request (in milliseconds). The “Max. Size Limit” is used to choose the maximum number of entries that should be returned. The “Synchronization Period” is used to determine the number of milliseconds between two synchronizations. The “Max. Threads” is used to determine the number of threads that should be used by the server in its execution.

A user can enter into the directory services by using Jxplorer that supports LDAPv3. For this purpose, the user must identify him by providing some security credentials. The screenshot of such scenario is described in figure 6.3 (e).
In our system, we have used a simple authentication. A user has to provide his corresponding distinguished name (DN) and password in order to enter into the directory service. He also needs to provide the server host name and port number. For the Apache Directory Server, the port number is 10389.

After entering into the directory service, a user can search for directory data, specifically books or other services. For example, figure 6.3 (f) explains that a user is looking for a book by providing documentTitle in the search filter.
Experimental Results and Performance Analysis

As the book is available, the user has obtained the book easily with some common information.

5.4 Performance Analysis

In the library system, LAN (Local Area Network) is used. The server computer and client computers are connected via the switch. The performance of LDAP server generally depends on the CPU capability, entry size and the number of entries. When the CPU is fast, the LDAP server can serve more client requests quickly. The size of entries and the number of entries also affect the performance of LDAP server. In the library system, Apache Directory Server is used as LDAP server. It was designed to be as fast as possible. It is easy with a fairly low-powered machine. It supports TLS and SSL for transport layer security. These two security techniques are used to encrypt data that transfer between client and server. Hence, data remains intact. This server can handle many clients at a time. In the library system, the server was checked serving 4 clients at a time and it works nice.

For searching any data, the Apache Directory Server used in the library system takes little time and the service is satisfactory. Moreover, by the help of this server, it is possible to update any entry quickly. It takes less time for the connection, processing and response.

Moreover, in the library system, the users can only search and read data. The librarians can update data. The administrators can manage the directory data and control the user access. No one can access directory services without providing authentication credentials. Therefore, the directory service provided by the library system is reliable.
Conclusions and Future work

6 Conclusions and Future work

The main purpose of our thesis work is to propose the architecture of a distributed LDAP directory. We have implemented this work as an application of a library system. In this system, the central library controls the data of its other subordinate libraries. For each library, there is an LDAP directory that contains different kinds of information. The central administrator of the central library has access to the other subordinate directories. So, the central administrator can provide information to the other subordinate administrators when needed. Moreover, he can import and export data. Hence, the data is shared and distributed among all directories in this system. The central administrator can control and manage directory services from a remote place.

In our library system, we have used the Apache Directory Server as an LDAP server. It supports LDAP version 3 and provides both SSL and TLS for security. Both SSL and TLS technologies provide strong security to the transport layer of the LDAP directory server. There are three categories of people in this system. They are administrators, librarians and users. Different categories of people have been given different access rights. The central administrator has rights to access all subordinate directories and to manage directory data. Other subordinate administrators have to control and manage only their own directory data. That is, no subordinate administrator has access rights to other directories. The librarians can only update their own directory data. A user enters his belonging library directory after providing many authentication credentials, namely a user password and a distinguished name. No one can enter into the system without providing an identification that is stored in the directory. Therefore, the system provides good security and access controls. Hence, the system is reliable.

In addition, the LDAP protocol provides easy access directory services. The user can easily search data from a remote network. Many users can access in the directory services at a time. LDAP operations are performed quickly. Also, the administrators can easily manage the directory data and control the user access staying at a remote network place.

In conclusion, it can be said that our distributed LDAP directory provides data distribution, reliability and better efficiency.

An LDAP directory provides tremendous facilities to the directory applications. It provides easy functionalities and data distribution. However, there is no standard authentication technique in an LDAP protocol. Therefore, in future, standard authentication techniques should be developed. As an LDAP directory provides data distribution, it can be applied to any distributed system where the data is needed to be distributed and stored.
References

7 References


Appendixes- Source Code

8 Appendixes- Source Code

In this chapter, we have included the client-server code for the socket communication in Java programming language that is used during our thesis implementation. The Appendix A contains the server side programming code and the Appendix B contains the client side programming code.

8.1 Appendix A

The server side code written in java is given below.

```java
import java.io.EOFException;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;

public class Server extends JFrame {
    private JTextField enterField; // inputs message from user
    private JTextArea displayArea; // display information to user
    private ObjectOutputStream output; // output stream to client
    private ObjectInputStream input; // input stream from client
    private ServerSocket server; // server socket
    private Socket connection; // connection to client
    private int counter = 1; // counter of number of connections
    // set up GUI
    public Server() {
        // code to set up GUI
    }
}
```


Appendixes - Source Code

```java
{ super ( "Central Library" );

enterField = new JTextField(); // create enterField
enterField.setEditable( false );
enterField.addActionListener(
    new ActionListener()
    {
        // send message to client
        public void actionPerformed( ActionEvent event )
        {
            sendData( event.getActionCommand() );
            enterField.setText( "" );
        } // end method actionPerformed
    } // end anonymous inner class
); // end call to addActionListener

add( enterField, BorderLayout.NORTH );
displayArea = new JTextArea(); // create displayArea
add( new JScrollPane( displayArea ), BorderLayout.CENTER );
setSize( 400, 250 ); // set size of window
setVisible( true ); // show window
} // end Server constructor

// set up and run server
public void runServer()
{
    try // set up server to receive connections; process connections
    {
        server = new ServerSocket( 12345, 100 ); // create ServerSocket

        while ( true )
        {
            try
```
Appendixes - Source Code

```java
{  
    waitForConnection(); // wait for a connection  
    getStreams(); // get input & output streams  
    processConnection(); // process connection  
} // end try  

catch ( EOFException eofException )  
{  
    displayMessage( "Central Library terminated connection" );  
} // end catch  

finally  
{  
    closeConnection(); // close connection  
    counter++;  
} // end finally  

} // end while  
} // end try

catch ( IOException ioException )  
{  
    ioException.printStackTrace();  
} // end catch  

} // end method runServer

// wait for connection to arrive, then display connection info  
private void waitForConnection() throws IOException  
{  
    displayMessage( "Waiting for connection\n" );  
    connection = server.accept(); // allow server to accept connection  
    displayMessage( "Connection " + counter + " received from: " +  
        connection.getInetAddress().getHostName() );  
} // end method waitForConnection

// get streams to send and receive data
private void getStreams() throws IOException
{
    // set up output stream for objects
    output = new ObjectOutputStream( connection.getOutputStream() );
    output.flush(); // flush output buffer to send header information

    // set up input stream for objects
    input = new ObjectInputStream( connection.getInputStream() );

    displayMessage( "\nGot I/O streams\n" );
} // end method getStreams

// process connection with client
private void processConnection() throws IOException
{
    String message = "Connection successful";
    sendData( message ); // send connection successful message

    // enable enterField so server user can send messages
    setTextFieldEditable( true );

    do // process messages sent from client
    {
        try // read message and display it
        {
            message = ( String ) input.readObject(); // read new message
            displayMessage( "\n" + message ); // display message
        } // end try
        catch ( ClassNotFoundException classNotFoundException )
        {
            displayMessage( "\nUnknown object type received" );
        } // end catch
    } // end do
} // end method processConnection
Appendixes - Source Code

} while ( !message.equalsIgnoreCase( "Local Library>>> BYE" ) );
} // end method processConnection

// close streams and socket
private void closeConnection()
{
    displayMessage( "\nTerminating connection\n" );
    setTextFieldEditable( false ); // disable enterField

    try
    {
        output.close(); // close output stream
        input.close(); // close input stream
        connection.close(); // close socket
    } // end try
    catch ( IOException ioException )
    {
        ioException.printStackTrace();
    } // end catch
} // end method closeConnection

// send message to client
private void sendData( String message )
{
    try // send object to client
    {
        output.writeObject( "Central Library>>> " + message );
        output.flush(); // flush output to client
        displayMessage( "\nCentral Library>>> " + message );
    } // end try
    catch ( IOException ioException )
{ } // end method sendData

// manipulates displayArea in the event-dispatch thread
private void displayMessage( final String messageToDisplay )
{
    SwingUtilities.invokeLater( 
        new Runnable()
        {
            public void run() // updates displayArea
            {
                displayArea.append( messageToDisplay ); // append message
            } // end method run
        } // end anonymous inner class
    ); // end call to SwingUtilities.invokeLater
} // end method displayMessage

// manipulates enterField in the event-dispatch thread
private void setTextFieldEditable( final boolean editable )
{
    SwingUtilities.invokeLater( 
        new Runnable()
        {
            public void run() // sets enterField's editability
            {
                enterField.setEditable( editable );
            } // end method run
        } // end inner class
    ); // end call to SwingUtilities.invokeLater
} // end method setTextFieldEditable
Appendixes - Source Code

} // end class Server

The main file is given below:
import javax.swing.JFrame;
public class ServerTest
{
    public static void main( String args[] )
    {
        Server application = new Server();
        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        application.runServer();
    }
}

8.2 Appendix B
The client side code written in java is shown below:
import java.io.EOFException;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.InetAddress;
import java.net.Socket;
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;
public class Client extends JFrame
Appendixes- Source Code

{
    private JTextField enterField; // enters information from user
    private JTextArea displayArea; // display information to user
    private ObjectOutputStream output; // output stream to server
    private ObjectInputStream input; // input stream from server
    private String message = ""; // message from server
    private String chatServer; // host server for this application
    private Socket client; // socket to communicate with server

    // initialize chatServer and set up GUI
    public Client( String host )
    {
        super( "Local Library" );

        chatServer = host; // set server to which this client connects
        enterField = new JTextField(); // create enterField
        enterField.setEditable( false );
        enterField.addActionListener( new ActionListener()
        {
            // send message to server
            public void actionPerformed( ActionEvent event )
            {
                sendData( event.getActionCommand() );
                enterField.setText( "" );
            } // end method actionPerformed
        }); // end call to addActionListener
        add( enterField, BorderLayout.NORTH );

        displayArea = new JTextArea(); // create displayArea
        add( new JScrollPane( displayArea ), BorderLayout.CENTER );
    }
}
setSize( 400, 250 ); // set size of window
setVisible( true ); // show window
} // end Client constructor

// connect to server and process messages from server
public void runClient()
{
    try // connect to server, get streams, process connection
    {
        connectToServer(); // create a Socket to make connection
        getStreams(); // get the input and output streams
        processConnection(); // process connection
    } // end try
    catch ( EOFException eofException )
    {
        displayMessage( "\nLocal Library terminated connection" );
    } // end catch
    catch ( IOException ioException )
    {
        ioException.printStackTrace();
    } // end catch
    finally
    {
        closeConnection(); // close connection
    } // end finally
} // end method runClient

// connect to server
private void connectToServer() throws IOException
{
    displayMessage( "Attempting connection\n" );
// create Socket to make connection to server
client = new Socket( "localhost", 12345 );
// display connection information
displayMessage( "Connected to: " +
    client.getInetAddress().getHostName() );
} // end method connectToServer

// get streams to send and receive data
private void getStreams() throws IOException
{
    // set up output stream for objects
    output = new ObjectOutputStream( client.getOutputStream() );
    output.flush(); // flush output buffer to send header information

    // set up input stream for objects
    input = new ObjectInputStream( client.getInputStream() );

    displayMessage( "\nGot I/O streams\n" );
} // end method getStreams

// process connection with server
private void processConnection() throws IOException
{
    // enable enterField so client user can send messages
    setTextFieldEditable( true );

    do // process messages sent from server
    {
        try // read message and display it
        {
            message = ( String ) input.readObject(); // read new message

            displayMessage( message );
        }
        catch( ClassNotFoundException e )
        {
            displayMessage( "Error: Class Not Found" );
        }
    }
} // end method processConnection()}
displayMessage( "\n" + message ); // display message
} // end try

catch ( ClassNotFoundException classNotFoundException )
{
    displayMessage( "\nUnknown object type received" );
} // end catch

} while ( !message.equalsIgnoreCase( "Central Library>>> BYE" ) );
} // end method processConnection

// close streams and socket
private void closeConnection()
{
    displayMessage( "\nClosing connection" );
    setTextFieldEditable( false ); // disable enterField

    try
    {
        output.close(); // close output stream
        input.close(); // close input stream
        client.close(); // close socket
    } // end try; // close socket

    catch ( IOException ioException )
    {
        ioException.printStackTrace();
    } // end catch
} // end method closeConnection

// send message to server
private void sendData( String message )
{
    try // send object to server
Appendixes- Source Code

```java
{ 
    output.writeObject( "Local Library>>> " + message );
    output.flush(); // flush data to output
    displayMessage( "\nLocal Library>>> " + message );
} // end try

catch ( IOException ioException )
{
    displayArea.append( "\nError writing object" );
} // end catch

} // end method sendData

// manipulates displayArea in the event-dispatch thread
private void displayMessage( final String messageToDisplay )
{
    SwingUtilities.invokeLater(
        new Runnable()
        {
            public void run() // updates displayArea
            {
                displayArea.append( messageToDisplay );
            } // end method run
        } // end anonymous inner class
    ); // end call to SwingUtilities.invokeLater
} // end method displayMessage

// manipulates enterField in the event-dispatch thread
private void setTextFieldEditable( final boolean editable )
{
    SwingUtilities.invokeLater(
        new Runnable()
        {
            public void run() // sets enterField's editability
            {
            } // end method run
        } // end anonymous inner class
    ); // end call to SwingUtilities.invokeLater
} // end method setTextFieldEditable
```
Appendixes - Source Code

```java
{  
    enterField.setEditable( editable );
} // end method run

} // end anonymous inner class

); // end call to SwingUtilities.invokeLater

} // end method setTextFieldEditable

} // end class Client

The main file is given below:

```java
import javax.swing.JFrame;

class ClientTest  
{
    public static void main( String args[] )
    {
        Client application; // declare client application

        // if no command line args
        if ( args.length == 0 )
            application = new Client( "127.0.0.1" ); // connect to localhost
        else
            application = new Client( args[ 0 ] ); // use args to connect

        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        application.runClient(); // run client application

    } // end main

} // end class ClientTest
```