DESIGN OF A GENERIC CLIENT-SERVER MESSAGING INTERFACE USING XML

by

Suvendi Chinnappen Rimer

Submitted in partial fulfilment of the requirements for the degree

Master of Engineering (Computer Engineering)

in the

Faculty of Engineering

UNIVERSITY OF PRETORIA

March 2003

DISSERTATION SUMMARY

DESIGN OF A GENERIC CLIENT-SERVER MESSAGING INTERFACE USING XML

by

Suvendi Chinnappen Rimer

Study Leader:	Prof. G. P. Hancke
Department:	Electrical, Electronic and Computer Engineering
	UNIVERSITY OF PRETORIA
Degree:	MEng (Computer Engineering)

Applications that use directory services or relational databases operate in client-server mode where a client requests information from a server, and the server returns a response to the client. Communication between each client-server application is achieved by using separate custom built front-ends with non-portable data formats. A need exists to access information from different heterogeneous client-server systems in a standard message request-response format.

This research proposes a generic XML document that presents a common request-response interface to the client from which they can access network protocol or database information. The XML component is easily adaptable to accessing any new client-server type protocol or database data that may be added to a server.

The approach in determining the XML elements is, firstly review each systems command and data structure separately, and then determine if there are any commonalities within each protocol that would allow for a common representation of both the data and command structure.

For the purposes of this project, three different data sources that are typically used in an Internet application were analysed, namely:

- A TCP based server program.
- A relational type database.
- A directory service.

The solution was implemented using Linux as the operating system, Java as the programming language, MySQL as the relational database, openLDAP as the directory server and a proprietary TCP based server application. Initially the complete system was developed for the proprietary TCP-based application. The other systems were added with minimum additional work.

The result of the implementation was that it is relatively easy to add new protocols (for e.g. LDAP) on an as needed basis with minimal changes required on the server side. A client will receive XML responses that the client can either adapt (typically using a separate style-sheet) to their specific needs or use the existing front-ends if they are suitable.

After the design was implemented and tested, the performance of XML and non-XML messages was evaluated. As expected the increased verbosity of XML results in a larger footprint that requires more processing time and resources. This means that any implementation using XML has to carefully weigh the benefits of flexibility, extensibility and standard message formats against reduced performance.

After evaluating XML type messages in an Internet type environment that involved human-computer interaction, it was concluded that the slower response times is not that significant to negate the benefits of a common message interface provided by using XML.

SAMEVATTING VAN VERHANDELING

DIE ONTWERP VAN 'N GENERIESE BOODSKAP-INTERVLAK VIR 'N KLIËNTBEDIENER-STELSEL MET BEHULP VAN XML

deur

Suvendi Chinnappen Rimer

Studieleier:	Prof G.P. Hancke
Departement:	Elektriese, Elektroniese en Rekenaar-Ingenieurswese
	UNIVERSITEIT VAN PRETORIA
Graad:	MIng (Rekenaar-Ingenieurswese)

Toepassings wat gidsdienste of verhoudingsdatabasisse gebruik, werk in 'n kliëntbediener-modus waar die klient inligting van die bediener versoek en die bediener 'n antwoord na die kliënt terugstuur. Kommunikasie tussen elke kliëntbediener-toepassing vind plaas deur die gebruik van afsonderlike toegewyde intervlakke met nie-oordraagbare data-formate. 'n Behoefte bestaan om toegang te verkry tot inligting vanaf verskillende heterogene kliëntbediener-stelsels deur 'n standaard versoek-antwoord-formaat te gebruik.

Hierdie navorsing stel 'n generiese XML-dokument voor wat 'n algemene versoek-antwoord-intervlak vir die kliënt aanbied waarvandaan kliënte toegang het tot netwerk-protokol- of databasis-inligting. Die XML-komponent is maklik aanpasbaar vir nuwe Internet-tipe protokolle of databasis-inligting wat tot die bediener bygevoeg word.

Die benadering tot die bepaling van die XML-elemente was om elke stelselopdrag en datastruktuur afsonderlik te beskou en dan te bepaal watter ooreenkomste binne die protokolle bestaan wat 'n gemeenskaplike voorstelling van beide die data- en die bevelstruktuur moontlik maak.

Vir die doel van hierdie navorsing is drie verskillende bronne van data wat tipies in Internet-toepassings gebruik word, ontleed:

- ‘n TCP-gebaseerde bediener-program
- ‘n verhoudingsdatabasis
- ‘n gidsdiens

Die oplossing is geïmplementeer deur Linux as die bedryfstelsel, Java as die programmeringstaal, MySQL as die verhoudingsdatabasis en openLDAP as die gidsdiens te gebruik. Eerste is die totale stelsel vir die TCP-gebaseerde toepassing ontwikkel en die ander twee toepassings is sonder veel bykomende werk bygevoeg.

Die bevinding tydens die implementeering was dat dit maklik is om nuwe protokolle (bv. LDAP) by te voeg met minimale veranderings aan die bediener se kant. ‘n Kliënt sal XML-antwoorde kry wat die kliënt kan aanpas vir sy spesifieke behoeftes of slegs deur die beskikbare intervlakke te gebruik.

Nadat die ontwerp geïmplementeer en getoets is, is die gedrag van XML teen nie–XML boodskappe geëvalueer. Soos verwag is, het die beter woordrykheid van die XML ‘n groter voetspoor tot gevolg wat meer prosesseertyd en hulpbronne benodig. Dit beteken dat enige implementering wat XML gebruik, die voordele van aanpasbaarheid, uitbreibaarheid en standaard boodskapformate teen verlaagde werkverrigting moet afspeel.

Nadat die gebruik van XML-tipe boodskappe beoordeel is in ‘n Internet-omgewing waar mens-rekenaar interaksie betrokke is, is tot die gevolgtrekking gekom dat die voordele van die gemeenskaplike boodskapintervlak, deur die gebruik van XML, groter is as nadeel van die effens stadiger reaksietyd.

List of Abbreviations

CIM	Common Information Model
DAP	Directory Access Protocol
DIT	Directory Information Tree
DN	Distinguished Name
DTD	Document Type Definition
DSML	Directory Services Markup Language
ESD	Extended Services Daemon
HTTP	HyperText Transport Protocol
LDAP	Lightweight Directory Access Protocol
OSI	Open Systems Interconnection
RDN	Relative Distinguished Name
SGML	Standard Generalized Markup Language
SOAP	Simple Object Access Protocol
SQL	Structure Query Language
TCP	Transmission Control Protocol
XML	eXtensible Markup Language
WSDL	Web Services Description Language
UDDI	Universal Description, Discovery and Integration

Table of contents

List of figures

Table of tables

Chapter 1 : RESEARCH OVERVIEW

1.1 Introduction

This chapter describes the current problems businesses are increasingly facing when using inter-networked computer systems to support multiple applications. It defines the problem statement, the research objectives, the research approach and the scope of the work undertaken.

1.2 Scope

The scope of the research is to provide a generic XML interface to multiple client server applications using Internet type transport protocols such as HTTP and TCP. It does not focus on any security concerns especially with respect to the transportation of data over an open network. It assumes that there will be some sort of access control mechanism in place on a client server application and therefore that some authentication information will have to be provided to the server applications.

1.3 Problem Statement

Protocols such as LDAP or applications that use a relational database operate using a client-server type model where the client requests information from a server, and the server returns a response to the client.

Applications using different protocols typically require support for each of the protocols to be implemented on a separate front-end component. Application interaction (via messages) is achieved by using separate custom built front-end applications with non-portable data formats and functionality.

Traditionally, a client connected to the Internet that requires access to different server applications would have protocol specific programs at the client to access the data. When a new server application is added to the system, a separate client application is developed to access the new protocol's data on the host server.

For example consider a typical system connected to the Internet that has the following applications/services running on it:

- A directory service (i.e. LDAP).
- A TCP based proprietary server program.
- A relational database server.

The traditional architecture would require separate client programs accessing each of the server or database applications as shown in the figure below.

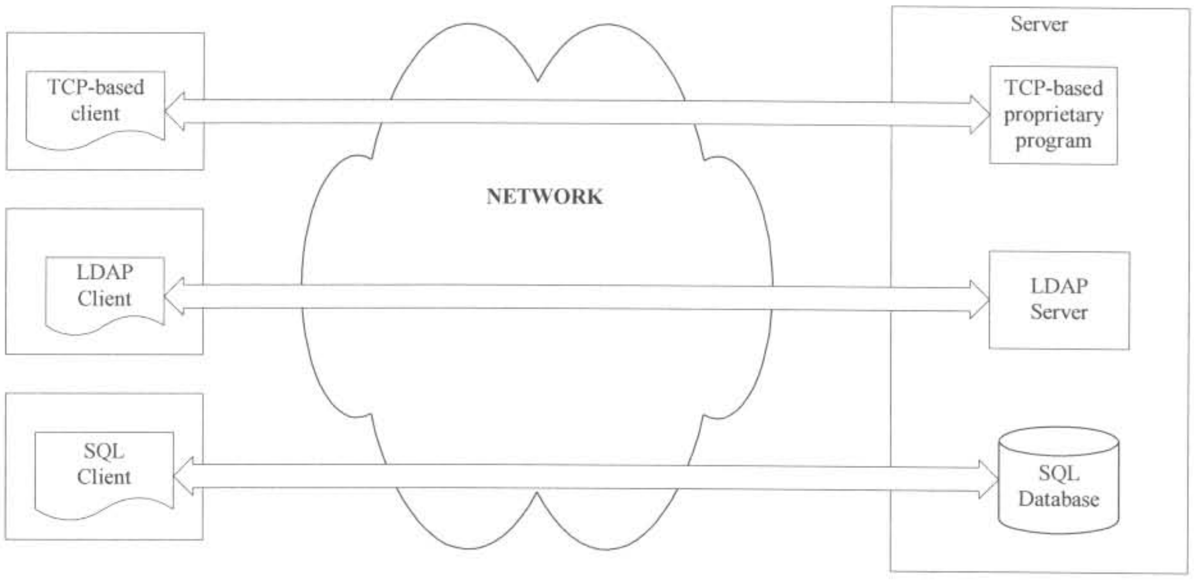


Figure 1: Traditional architecture requiring separate client applications

The system architecture depicted in Figure 1 increases the complexity on the client's side, as it needs to have the latest version of each specific protocol program installed on its workstation. In addition, if a new server application is added, each client needs to load another client program that can access the server's data. The need to install specialised clients on workstations increases the complexity of maintaining a system and reduces flexibility to introduce new protocols into a system. Any addition of new software may require the addition of new-shared libraries that may complicate or interfere with existing applications (such as stability, versions etcetera).

A need exists to access information from different heterogeneous systems in a standard message request-response format. The rules surrounding request-response type messages and their data should be organized in a clear and consistent manner, so that information can be shared among many applications.

The eXtensible Markup Language (XML) is proposed as a solution to access diverse information systems through a common metadata model. XML is particularly suited to web-based data exchange because it allows data exchange across disparate platforms and operating systems.

This project describes an XML model that abstracts the differences in underlying heterogeneous client-server systems and provides a common XML message interface.

To design the model the similarities of different client-server applications were identified, and using these similarities a common messaging system using XML as the command interpretation language was developed.

Figure 2 provides a high-level overview of the proposed system architecture.

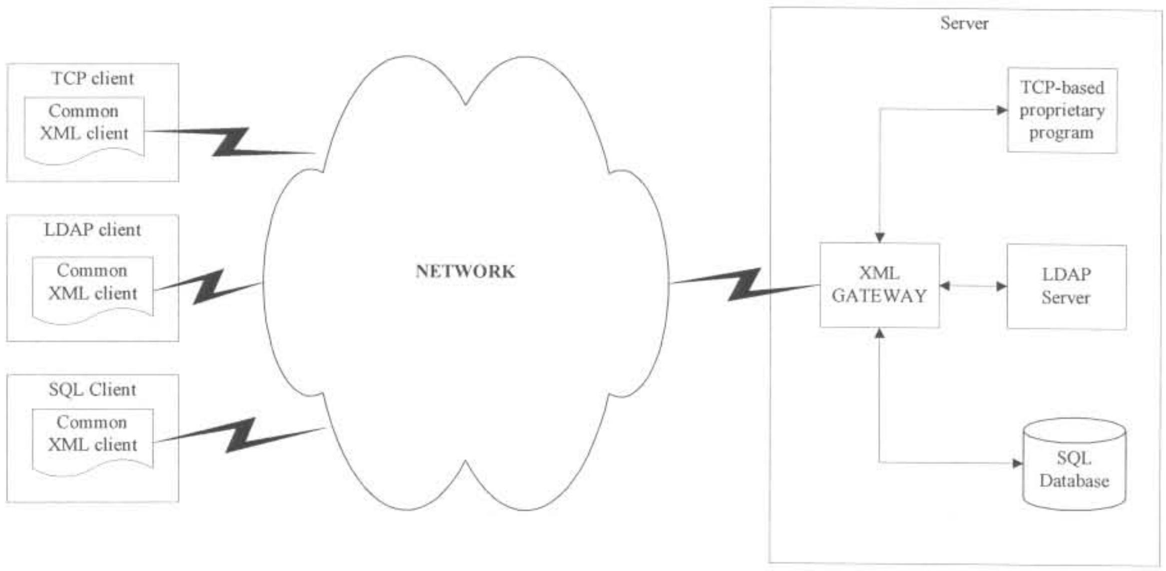


Figure 2: Proposed architecture depicting protocol-to-XML clients interfacing to an XML gateway

1.4 Research Context

Studies on the problem of restructuring and reformatting of data as it passes from one software tool or process to another predates the widespread use of the Internet that started in the mid 1990s. Blattner et al. [18,19] in a study on generic message translation, attempted to solve the problem by providing a visual interface that can create a mapping between fields in different message types that specifies which fields have similar semantic content. The Blattner et al. papers were published before the introduction of XML into the computing landscape. However, in their paper, the authors conclude that some sort of "parser-generator" must be constructed to take descriptions for data specifications and create a "translator" between systems.

Since the introduction of XML, several studies have been undertaken on the feasibility of using XML, such as the article by Bi et al. [15] that focuses on using XML to interact with multiple legacy applications, and an article by Peinl and Mitchang [20], which investigates transforming independent, autonomous data sources into a common XML format in order to provide an integrated communication platform for mobile applications. Both articles acknowledge the advantages provided by using XML as the data modelling and exchange mechanism between applications (clients) and information sources (servers).

However, use of XML for generic messaging does not come without some disadvantages, namely slower processing speed caused by the additional overhead of using XML to transform and parse messages. Boedjang et al. [21] in their study of distributed data structures conclude that the performance measurements of applications that run application-specific code are faster than those that use generic message passing software.

1.5 Research Objective

The main objective of this research is to abstract the differences in underlying heterogeneous systems by designing a generic XML component that provides a common client that can interface with multiple heterogeneous client-server applications.

To achieve this, a number of sub-objectives were examined. These sub-objectives are:

- Examine the current process in which the individual client-server applications (such as

LDAP, SQL and ESD) interact,

- Investigate current implementations of XML-LDAP and XML-SQL,
- Determine XML strengths and weaknesses,
- Identify advantages, if any in using XML for a common messaging system for multiple heterogeneous applications,
- Investigate methods of error handling within the XML gateway component,
- Investigate using SOAP as the method of communicating with both client and server devices and with other management entities,
- Identify problems in providing a single messaging interface to multiple applications,
- Analyse any existing generic XML interfaces that may be similar to area of research undertaken,
- Investigate the current functions and objects used in the existing applications that will have to be implemented in the XML gateway,
- Develop a generic XML schema that correctly models the commands and data of the different applications.
- Design and develop a generic XML front-end that allows current Internet protocols to access the gateway services through the same application, and
- Design and develop a generic XML gateway for use by multiple client-server applications.

1.6 Research Approach

This section identifies the main questions that need to be answered and describes the research approach used in attempting to answer these questions.

1.6.1 Research Questions

The following problem solving questions were identified to assist in better understanding and defining the problem.

1. What is the current means of communication in terms of command structure and message format between each of the multiple client-server applications?
2. What commonalities do these diverse applications share in terms of command structure

and data format?

3. What advantages will be derived from using XML as the common interface language?
4. What disadvantages will arise from using a single XML interface to multiple applications?
5. What are the application boundaries?
6. Why use XML?
7. How will the various technologies and applications interact?
8. What are the system resource constraints (if any) on the application?
9. How will the XML gateway application decide which of the different applications to send a request to?
10. Does XML provide sufficient benefit to overcome its shortcomings?

1.6.2 Research Instruments

1. Literature Study:
   a. The scope of the work was identified.
   b. A literature study was undertaken to understand each applications message and data structure used in client-server interaction.
   c. A literature study was undertaken to investigate existing XML standards (if any) for LDAP and SQL. Note ESD is a proprietary protocol and there is no previous XML framework/standard for it.
   d. A literature study was undertaken to understand the structure and syntax of XML.
   e. A literature study was undertaken to understand the structure and syntax of the XML DTD and the XML Schema.
   f. A search was done to identify appropriate open source applications that would serve as an LDAP server, a relational database, a web server and an XML parser.

2. Problem Solving Analysis
   a. An analysis of each of the applications message and data structure was undertaken and certain commonalities in commands and parameters were

identified.

b. An analysis of existing LDAP-XML and SQL-XML frameworks was conducted to determine if they were suitable for the needs of the study/research problem.

c. An analysis of XML DTD's and XML schema was undertaken to determine which would be the most appropriate to define the designed common data model.

3. Design
    a. An XML schema was developed that models the request-response command structures of client server applications in a generic way that can be used across multiple client-server type applications.
    b. An addressing mechanism was devised to identify which application the request is destined for.
    c. An XML gateway was designed to send requests to the correct destination application using a structure that allows for easy addition of new applications.
    d. A front-end interface was designed to provide a GUI to users to send requests to multiple applications and to view the corresponding responses in a user-friendly interface.
    e. Several test programs were written to test individual components of the design.

4. Implementation
    a. The open source applications that could serve as an LDAP server, a database, a web server and an XML parser were set up to run on the specified operating system.
    b. The design was implemented using an appropriate programming language.
    c. The design was tested using several test programs.
    d. The complete design was tested using the implemented web front-end component.

5. Analysis and Assessment
    a. A final analysis of the system in terms of, meeting functional requirements, and

performance was undertaken.

b. The results were provided and an analysis of the results in terms of the above-mentioned research objectives and questions is provided.

c. Final conclusions of the results are provided.

Chapter 2 : LITERATURE STUDY

2.1 Overview of application technology

This chapter describes the server applications used in the design and implementation of the research objectives. It provides a brief overview of a directory service, namely the Lightweight Directory Access Protocol (LDAP), a proprietary socket service protocol, i.e. the Extended Services Daemon (ESD) and SQL and database theory. These three "protocols" are examined because they are used in the design and implementation stage to verify that a common XML model is flexible enough to send and receive messages to heterogeneous server applications.

In addition, a brief overview of XML is provided and the two main current standard methods of creating an XML document, namely the XML document type definition or the XML schema.

A brief look at the attempts made to standardize XML and LDAP and XML and SQL is provided. A short description of the Common Information Model and SOAP is also provided, together with reasons why it was not used. Finally an overview of current research published in articles that are related to this area of research is given.

2.1.1 Lightweight Directory Access Protocol (LDAP)

2.1.1.1 Introduction: LDAP

The X.500 standard was created by the CCITT in 1988 to (among other functionality) specify the communication between the directory client and the directory server using the directory access protocol (DAP). DAP is an application layer protocol and requires the entire OSI protocol stack to operate. The disadvantage of using the OSI protocol stack is that it requires more resources than are available in many small environments.

LDAP was developed as a lightweight alternative to DAP. LDAP requires the less weighty TCP/IP protocol stack and uses a simplified subset of X.500 operations. LDAP defines the communication protocol, i.e. the transport and format of messages used by a client to access data in an X.500-like directory but it does not define the directory service itself.

2.1.1.2 Overview of LDAP Architecture

LDAP is a client-server system. The server can use a variety of databases to store a directory, each optimised for quick and copious read operations. When an LDAP client application connects to an LDAP server it can either query a directory or upload information to it. In the event of a query, the server either answers the query or, if it cannot answer locally, it can refer the query upstream to a higher-level LDAP server that does have the answer. If the client application is attempting to upload information to an LDAP directory, the server verifies that the user has permission to make the change and then adds or updates the information.

The content of messages exchanged between an LDAP client and an LDAP server is defined by LDAP. The messages specify the operations requested by the client (search, modify, delete, and so on), the responses from the server, and the format of data carried in the messages. LDAP messages are carried over TCP/IP, a connection-oriented protocol; so there are also operations to establish and disconnect a session between the client and server [1, 26, 27].

The general interaction between an LDAP client and an LDAP server takes the following form:

- Client attempts to establish a connection to the server. To do this, the client will require the host name or IP address and TCP/IP port number where the LDAP server is listening. If authentication is required, the client also needs to provide a user name and a password.
- If a connection is successfully established, the client can send messages to perform operations on the specified directory data.
- When the client has completed all requests, it closes the connection with the server.

2.1.1.3 Operations supported in LDAP

A directory is a listing of information about objects arranged in some order that gives details about each object. The directory stores and organizes data structures known as entries.
A directory entry usually describes an object such as a person, a printer, a server, and so on. LDAP defines the following operations for accessing and modifying directory entries [1, 26]:

- Searching for entries meeting user-specified criteria
- Adding an entry
- Deleting an entry
- Modifying an entry
- Modifying the distinguished name or relative distinguished name of an entry (move)
- Comparing an entry

2.1.1.4 The LDAP Models

LDAP is based on four models: Information, Naming, Functional and Security models.

The Information, Naming and Functional models are relevant to the research conducted and are discussed in more detail in the following sections. The security aspect was not relevant to the research and is not discussed.

1. The Information Model

The information model describes the structure of information stored in an LDAP directory. The basic unit of information stored in the directory is called an entry. Entries represent objects of interest in the real world such as people, servers, organizations, and so on. Entries are composed of a collection of attributes that contain information about the object. Every attribute has a type and one or more values and syntax. The syntax specifies what kind of values can be stored. In addition to defining what data can be stored as the value of an attribute, an attribute's syntax also defines how those values behave during searches and other directory operations. The relationship between a directory entry and its attributes and their values is shown in Figure 3 [1, 26].

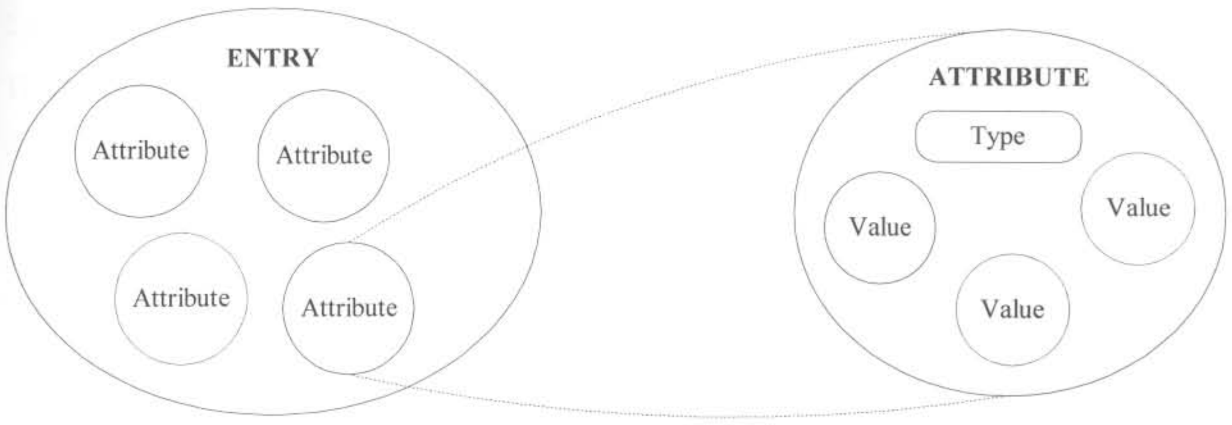


Figure 3: Entries, Attributes and Values [1]

2. The Naming Model

The LDAP naming model defines how entries are identified and organized. Entries are organized in a tree-like structure called the Directory Information Tree (DIT). Entries are arranged within the DIT based on their distinguished name (DN). A DN is a unique name that unambiguously identifies a single entry. DNs are made up of a sequence of relative distinguished names (RDNs), separated by commas. Each RDN in a DN corresponds to a branch in the DIT leading from the root of the DIT to the directory entry. DNs are used as primary keys to entries in the directory. LDAP defines a user-oriented string representation of DNs [1].

The Functional Model

The functional model describes what operations can be performed on the information stored in an LDAP directory. LDAP defines operations for accessing and modifying directory entries. LDAP operations can be divided into the following three categories:

- Queries include the search and compare operations used to retrieve information from a directory.
- Update includes add, delete, and modify operations used to update stored information in a directory.
- Authentication includes the bind, unbind, and abandon operations used to connect and disconnect to and from an LDAP server, establish access rights and protect information.

2.1.1.5 LDAP Operations

The following operations are used in the project implementation and are discussed in more detail.

Search

The search operation allows a client to request that an LDAP server search through some portion of the DIT for information meeting user-specified criteria in order to read and list the result(s). There are no separate operations for read and list; they are incorporated in the search function. The search can be very general or very specific. The search operation allows one to specify the starting point within the DIT, how deep within the DIT to search, what attributes an entry must have to be considered a match, and what attributes to return for matched entries.

Update, Add and Delete Operations

The operations used in the implementation are summarized below:

- Update: modify the contents of the directory.
- Add: inserts new entries into the directory
- Delete: deletes existing entries from the directory. Only leaf nodes can be deleted.

2.1.1.6 LDAP Data Interchange Format (LDIF)

The LDAP Data Interchange Format (LDIF) is used to import and export directory information between LDAP-based directory servers. The LDIF format is used to convey directory information or a description of a set of changes made to directory entries. An LDIF file consists of a series of records separated by line separators. A record consists of a sequence of lines describing a directory entry or a sequence of lines describing a set of changes to a single directory entry. An LDIF file specifies a set of directory entries or a set of changes to be applied to directory entries, but not both at the same time.

The basic form of a directory entry represented in LDIF is:

```
[<id>]
dn: <distinguished name>
objectClass: <object class>
objectClass: <object class>
```

```
...
<attribute type>[;language tag]:<attribute value>
<attribute type>[;language tag]:<attribute value>
...
```

Only the DN and at least one object class definition are required. In addition, any attributes required by the object classes for the entry must also be defined in the entry. All other attributes and object classes are optional. You can specify object classes and attributes in any order. The space character after the colon is optional [1].

2.1.2 Databases and SQL

A database is a structured collection of data. A relational database is built of entities and relationships. A relational database stores data in separate tables. The tables are linked to each other by defined relations. This makes it possible to combine data from several tables on request. A database table consists of a set of columns and rows represented in the following structure:

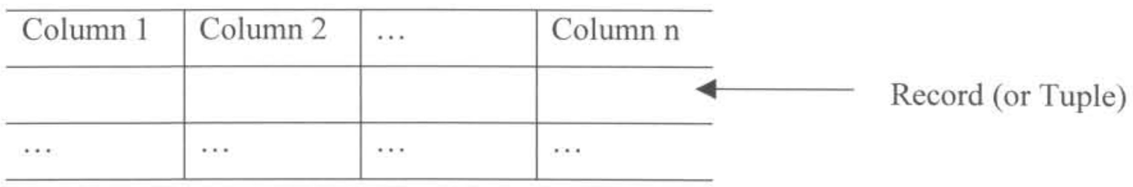


Figure 4: Database table structure [2]

Each table in a database has a unique name. A table consists of rows that contain the stored information. Each row contains exactly one record (or tuple). A table can have one or more columns. Each column name represents a specified field that has a specified data type that describes an attribute of the records. A table's structure (or relation schema) is defined by its attributes. A database schema is a set of relation schemas. The extension of a database schema at database run-time is called a database instance or database, for short.

The typical field (data) types used in database tables are shown in the table below [2].

Field Type	Description	Example
VARCHAR	Variable-length character string. Only the bytes used for a string require storage	varchar (80)
CHAR	Fixed-length character data, n characters long. Strings of type char are always padded with blanks to full length of n	char (40)
NUMBER	Numeric data type for integers and real	number (8),number (5,2)
INT	Stores values of type integer	int(4)
FLOAT	Stores values of type float	Float
DATE and TIME	Stores date and time values	'1-Jan-03 12:30:02'

Table 1: Database field types

As long as no constraint restricts the possible values of an attribute, it may have the special value null (for unknown). This value is different from the number 0,and it is also different from the empty string ''. Further properties of tables are:

- the order in which records appear in a table is not relevant (unless a query requires an explicit sorting).
- a table has no duplicate records (depending on the query, however, duplicate records can appear in the query result).

The Structured Query Language (SQL) is a standard for data manipulation first established by ANSI in 1982. SQL instructions are used to control relational databases, i.e. SQL instructions are used to access, define and manipulate the data in a relational database. The SQL used in this implementation is ANSI or standard SQL and only relevant areas of the querying language are discussed.

Select

We only required the use of simple SQL queries. A simple SQL query has the following (simplified) form (components in brackets [] are optional):

```
select [distinct ] <column(s)>
from <table>
[where <condition> ]
[order by <column(s) [asc |desc ]> ]
```

The columns to be selected from a table are specified after the keyword select.

Insert

The insert statement is used to insert records into a table. A simple SQL insert statement has the following format.

```
insert into <table> [(< column i,...,column j >]
values (<value i,...,value j >;
```

For each of the listed columns, a corresponding (matching) value must be specified. An insert does not necessarily have to follow the order of the attributes as specified in the creation of the table. If a column is omitted, the value null is inserted instead [2].

Update

To modify attributes of a record, the update statement is used. A simple SQL update statement has the following format.

```
update <table> set
<column i > =< expression i >...,<column j > =<expression j >
[where <condition>;
```

An expression consists of a constant (new value), an arithmetic or string operation, or a SQL query. Note that the new value to assign to <column i > must be a matching data type. If an update statement does not have a where clause, all the specified attributes of all records in the specified table are changed [2].

Delete

Records can be deleted from a table using the following delete command:

```
delete from <table> [where <condition>];
```

If the where clause is omitted, all records are deleted from the table [2].

2.1.3 Differences Between Directories and Databases

Important differences between directories and general-purpose databases are:

Directories	**Databases**
Optimized for read access because they are accessed (read or searched) much more often than they are updated (written).	Need to support applications (such as airline reservation and banking) with high update volumes.
May not support transactions	Always support transactions.
LDAP directories use a simplified and optimized access protocol that can be used in slim and relatively simple applications	Most databases support a standardized access method (SQL, that allows complex update and query functions at the cost of program size and application complexity
Directories are not intended to provide as many functions as general-purpose databases,	

Table 2: Differences between directories and databases

2.1.4 The IGUANA gateway

The IGUANA gateway provides access between fieldbus networks and clients using certain Internet protocols. The gateway uses the FEB protocol to communicate with fieldbus networks. The fieldbus system is a master–slave system, where the gateway is the master and all other nodes on the fieldbus are slaves. Only the gateway can actively initiate communication, slaves can only wait for the master to query them.

The gateway retrieves data from nodes on a fieldbus network. This data is represented as data points. The gateway provides access to a data point to clients using the Internet to connect to the gateway. The gateway allows a client connected to the Internet to read and write data to nodes on the fieldbus.

Besides this data point access, the gateway provides the following services:

- Asynchronous notifications, i.e. inform the user about special situations that have occurred by sending email, SMS messages or SNMP traps.
- Log the values of the data point and store these logs.

These services are available within a protocol called the Extended Services Daemon (ESD) protocol. This protocol is text-based and uses TCP. It works as a stateless request-response protocol. Multiple clients can simultaneously connect to the gateway and execute commands. The gateway routes these commands to the specified node on a fieldbus network. Not all commands require access to fieldbus network [3,4].

2.1.5 Extended Service Daemon (ESD)

ESD is a proprietary protocol used by the gateway to access data on the gateway, different fieldbus networks or nodes. ESD uses TCP to establish a connection between a client and the server. It uses request-response type commands with the ESD server acting as the slave and only responding when the master (a client) initiates a request.

The message protocol requires that all messages be terminated by a CR-LF (carriage return - line feed) pair. The maximum size of a message is 2048 characters, including spaces and the CR-LF pair. All responses contain a response code. A response code of zero indicates no error has occurred. A response code greater than zero indicates an error has occurred. No negative values are used. For multiple line responses, the end of a message is indicated by a line containing only a period (.) and a CR-LF pair.

The ESD protocol specifies a “data point” as an object that stores a value somewhere on the fieldbus network. To access the data point, a user needs to specify the field area network (FAN), the FAN daemon id, the node address and the data point address.

ESD Addressing

The FAN daemon id comprises the FAN type, which specifies the type of fieldbus network the FAN daemon connects to and a unique identifier, separated by a dot character (i.e. FANTYPE.FANID).

A node on a FAN is identified by a node address. A node address comprises two parts, namely the FAN daemon id as discussed previously and the FAN specific physical node address separated by an exclamation mark (i.e. FANId!FanNodeAddress).

The data point address identifies a data point at a node on a specific field area network. It consists of the FAN daemon id, the data point id and the physical node address in the format FANId!Dpid@FanNodeAddress [3].

Differentiation between FAN types in ESD is done in the FAN name. It is split into two parts:

- the FAN type and
- the name of the FAN (which is unique in every FAN family).

For example a valid FAN name is LON.MARIA where MARIA is one member of the LON FAN family.

A node is any kind of device that is attached to the FAN and each node has got a collection of data points it exposes to the FAN and ESD. Each data point basically has a name, a data type and a value that can be read and changed.

The main purpose of events is to record a data point's value in a defined time raster or on some special incident. Recorded data of events can be read out in logs. Each log has a timestamp that tells when it was taken and the data point's value [5].

Each data point has a value and an encoding. The encoding represents the type of the data value, i.e., integer that is equivalent to SCALARBIN in ESD notation or sequence of bytes, which is equivalent to BINHEX in ESD notation [3,4].

An event can be registered with ESD. Event criteria have to be specified for each event. The event criterion defines the conditions under which the event is triggered. For details about the event criteria and actions available please refer to reference [3].

The commands available from the ESD protocol are described in the Table 3 [3].

Request Command	Request Description	Request Parameters	Response	Error Response
READ	Requests the value of the specified FAN data point in the specified encoding style.	Data point address and encoding.	Numeric response code (zero), the data point address, encoding style and data point value.	Numeric response code (non zero) and error description.
WRITE	Updates the value of the specified data point for the specified encoding style.	Data point address, encoding and new data point value.	Numeric response code (zero) and description (OK).	Numeric response code (non zero) and error description.
DPINFO	Retrieves information about a specified data point	Data point address.	Numeric response code (zero), the data point address, data type of data point, set of encodings supported for data point, access rights to data point, data point name and data point self identification string.	Numeric response code (non zero) and error description.
NODEINFO	Retrieves information about a particular FAN node.	Node address.	Numeric response code (zero), node address, number of data points on this node, node self-identification string, encoded node location [optional], and encoded node program ID [optional].	Numeric response code (non zero) and error description.

i1734413X
916338571

Request Command	Request Description	Request Parameters	Response	Error Response
ENODEINFO	Retrieves information about a specified FAN node and all of the data points that belong to this node.	Node address.	Numeric response code (zero), node address, number of data points on this node, node self-identification string, encoded node location [optional], and encoded node program ID [optional] and the data point information (described in dpinfo's response for every data point on the node).	Numeric response code (non zero) and error description.
NODELIST	Requests ESD for a complete list containing all nodes of all FANs.	None	Numeric response code (zero), nodelist sequence number and list of all node addresses.	Numeric response code (non zero) and error description.
NODELISTSEQNO	Requests the sequence number of the nodelist.	None	Numeric response code (zero), and nodelist sequence number.	Numeric response code (non zero) and error description.
REFRESHNODELIST	Forces ESD to update its nodelist using the cached nodelists of the FAN daemons.	FAN daemon ID [optional]	Numeric response code (zero) and description (OK).	Numeric response code (non zero) and error description.

Request Command	Request Description	Request Parameters	Response	Error Response
UPDATENODELIST	Forces ESD to update the contents of its nodelist and forces FAN daemons to update their cached nodelists.	FAN daemon ID [optional]	Numeric response code (zero) and description (OK).	Numeric response code (non zero) and error description.
CREATEEVENT	Requests ESD to register an event with the given parameters.	Event criteria, event action, event description and event ICC private parameters [optional].	Numeric response code (zero) and event id assigned to event by ESD.	Numeric response code (non zero) and error description.
DELEVENT	Requests ESD to delete the specified event.	Event Id	Numeric response code (zero) and description (OK).	Numeric response code (non zero) and error description.
EVENTLIST	Requests ESD for a complete list of registered events.	None	Numeric response code (zero), event list sequence number, followed by a list of all the events in the order of event id, event action, event description and event parameters.	Numeric response code (non zero) and error description.

Request Command	Request Description	Request Parameters	Response	Error Response
EVENTLISTSEQNO	Requests ESD for the actual sequence number of the event list	None	Numeric response code (zero), and event list sequence number.	Numeric response code (non zero) and error description.
LOG	Requests ESD for a complete list of log entries associated with a specified event id.	Event Id	Numeric response code (zero) and description (OK), followed by a list of all log entries for the specified event id, in the form of: log line number of this log entry, the timestamp and the log entry data.	Numeric response code (non zero) and error description.
VERSION	Reports the current ESD application's version number.	None	Numeric response code (zero), and the version string.	None specified.

Table 3: ESD request-response commands

2.1.6 XML

The following sections provide a brief overview of XML. This is not intended as a full description of XML. For more information, refer to http://www.w3.org/XML/, specifically references [22, 23].

2.1.6.1 XML Introduction

The eXtensible Markup Language (XML) is a simplified sub-set of the Standard Generalized Markup Language (SGML). XML is a meta-language framework for defining and using tag-based markup languages.

It uses a portable format that is both machine and human readable, and is used to produce documents that convey content with semantic structure. An XML document does not need to be a file; it can be generated dynamically.

An XML document consists of text data and markup tags. As with HTML, data is identified using tags (identifiers enclosed in angle brackets, like this: <...>). Collectively, the tags (also called elements) are known as "markup". The markup indicates the syntactical structure of the document. However, unlike HTML, XML elements *identify* the data, rather than specifying how to display it [7].

The process of identifying data, called metadata provides some sense of how to interpret it. Therefore, XML can be thought of as a mechanism for specifying the *semantics* (meaning) of the data.

XML elements can be defined in a similar manner to the field names for a data structure, i.e., names are given that make sense for a given application. If multiple applications use the same XML data, all applications need to use the same element names.

XML elements can also contain attributes. Attributes are additional information included as part of the element itself, within the element's angle brackets. An attribute name is followed by an equal sign and the attribute value, and multiple attributes are separated by spaces.

A general XML requirement is that the elements of an XML document must form a tree and the tree structure of elements must be clearly shown in markup. The parent-child tree relationship corresponds to how elements are nested within each other in the linear text. The following tree conditions must be met: [6].

- There must be a root of the element tree that contains all other elements.
- Start and end tags must be properly nested – overlapping elements are not allowed.
- All elements, including empty elements (i.e. elements which contain no value), must have both the start and end tag.

2.1.6.2 XML Parsers

XML parsers are used to process XML. A parser is a software component that sits between the application and the XML files. It interprets the XML file and creates the document tree. It may also check the document's syntax and structure. There are two primary ways that applications can obtain information from an XML parser: as an event stream or as an object-based tree interface. The following figure shows the mutual relationship between the XML document, the XML parser and the application.

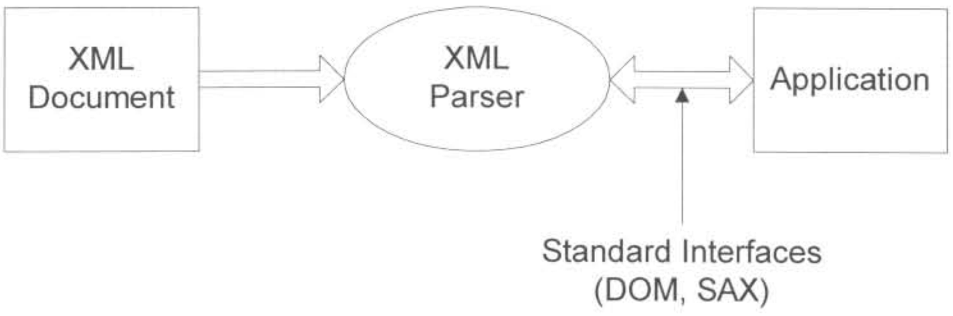


Figure 5 : XML data, the parser, and the application [6]

The application goes through the content of the XML file through the parser. The parser and the application must share a common model for XML data.

Using an object-based interface, a parser explicitly builds a tree of objects that contains all the elements in the XML document. The application is handed a tree in memory that exactly matches the XML document (file). The Document Object Model (DOM) provides an API for representing the logical structure (a tree) of documents.

With an event-based interface, the parser does not explicitly build a tree of objects. Instead, it reads the file and generates events as it finds elements, attributes, or text in the file. There are events for element start, element end, attributes, etcetera. An event-based interface tends to be more efficient because it does not explicitly build the XML tree in memory. Fewer objects are required and less memory is used [8].

The Simple API for XML (SAX) is the de facto standard for event-stream support. It provides a callback API so that an application can be notified of XML elements as they are parsed.

2.1.6.3 XML file structure

The beginning of any XML file must contain the following processing instruction that identifies the document as an XML document:
<?xml version="1.0"encoding="ISO-8859-1"standalone="yes"?>

The attributes in the heading are:

- **Version:** Identifies the version of the XML markup language used in the data. This attribute is not optional.
- **Encoding:** Identifies the character set used to encode the data. "ISO-8859-1" is "Latin-1" the Western European and English language character set. (The default is compressed Unicode: UTF-8).
- **Standalone:** Tells whether or not this document references an external entity or an external data type specification. If there are no external references, then "yes" is appropriate.

2.1.6.4 XML Advantages

- The interpretation of XML languages is unconstrained by XML itself, i.e. XML can be used for describing data, programs that process data, communication protocols or transmitting data, etcetera. The interpretation of an XML language is entirely up to the application that uses it.

- It is relatively easy to switch between the character sequence of XML and the tree-structured data view of XML, using an XML parser.
- Character sequences are easy to send over the network using standard protocols.
- Tree-structured data is easy to work with and it is easy to transform one tree into another.
- Since XML can encode both data and metadata, applications that communicate using XML can discover each other and establish a communication channel without prior arrangements.
- As XML is text-based, a standard text editor can be used to create and edit files.
- XML tells you what kind of data you have, not how to display it. This means that different parts of the information can be used in different ways by different applications. For example, an email program can process it, a search program can look for messages sent to particular people, and an address book can extract the address information from the rest of the message.
- XML provides a way to validate the data being delivered across the Internet, thus verifying that all data is complete.
- Validation of the XML message structure is inherent in the structure of the message.

2.1.6.5 XML Disadvantages

- Difficulties in using embedded binary data. The following techniques were proposed for embedding binary data in XML documents [13];
  - Refer to the data externally (such as with a URL)
  - Represent the data with MIME
  - Embed the binary data as CDATA

  However, CDATA cannot truly contain binary data as the CDATA end marker "]]>" is merely an unusual sequence, not a forbidden one.

- Larger footprint (message size): XML combined with associated standards is more verbose than standard protocols or languages such as SNMP, LDAP and SQL and should have higher static and dynamic implementation footprints.

However, the benefits from using XML as the standard messaging mechanism between the application and its external client interface is simpler with XML, and may result in an overall reduction in footprint size.

2.1.6.6 XML DTD

A Document Type Definition (DTD) is a mechanism to describe every object (element, attribute etc.) that can appear in a document. The DTD also defines the hierarchical structure of an XML document, including the order in which the elements must occur. For example, the following is a DTD element declaration:

```
<!ELEMENT address-book (entry+)>.
```

In the above example, the right side (the content model) defines the left side (the element name), i.e. the content model lists the children that are acceptable in the element. In the example this indicates that the address-book element contains one or more entry elements [8].

The following table describes the typical components of the DTD syntax.

Name	Description
<!ELEMENT >	Contains the element name followed by its content model.
#PCDATA	Parsed character data implying the element can contain text.
EMPTY	Means the element is an empty element
ANY	Means the element can contain any other element declared in the DTD.
<!ATTLIST …>	Defines the elements attributes. The components consist of the element name, the attribute name, the attribute type and a default value.
CDATA	Raw character data.

Table 4: DTD Syntax

A DTD is used to validate a document's XML structure, because it specifies which elements are allowed where in the XML document. The DTD can exist at the front of the document, as part of the prolog. It can also exist as a separate entity (external file), or it can be split between the document prolog and one or more additional entities.

The DTD syntax is different from XML document syntax. There are also difficulties in specifying a DTD for a complex document in such a way that it prevents all invalid combinations and allows all the valid ones. Also, DTDs are not very effective in specifying data ranges and have limited capability in specifying data types, i.e. DTDs support at most 10 data types.

2.1.6.7 XML Schema

XML Schema is the W3C schema specification for XML documents. An XML schema is a vocabulary for expressing a set of data's business rules. XML Schemas have a similar purpose as DTDs, namely to specify the following:

- the *structure* of XML documents, (for e.g. this element contains these elements, which contains these other elements, etcetera).
- the *data type* of each element/attribute (for e.g. this element shall hold an integer with the range 0 to 12,000).

The XML Schema is a significantly more powerful language than DTD. The following are some of the advantages of XML Schemas [7, 8, 23, 28]:

- Uses the same syntax as XML documents
- Improves data typing to support strings, and also numbers, dates etcetera.
- Supports more than 44 different data types.
- Allows you to create your own data types and to extend or restrict a type (derive new types on the basis of old ones).
- Introduces object-oriented concepts such as inheritance.
- Can express sets, i.e., can define the child elements to occur in any order.
- Can define multiple elements with the same name but different content.

- Can define elements with nil content.
- Can define substitutable elements - e.g., the "Book" element is substitutable for the "Publication" element.

However, XML schemas are generally more verbose than equivalent DTDs.

2.1.6.8 XML Schema Syntax

Declaring Elements

There are three patterns for declaring elements [6, 23, 28]:

- With a named type, specified by the type attribute, for example:

```
<xs:element name="name" type="type" minOccurs="int" maxOccurs="int" />
```

    - Examples of `type` are int, string etcetera.
    - minOccurs and maxOccurs are the minimum and maximum number of times the element can occur respectively.

- With an unnamed complex type, for example:

```
<xs:element name="name" minOccurs="int" maxOccurs="int" />
     <xs:complexType>
     …
     </xs:complexType>
</xs:element
```

- With an unnamed simple type, for example:

```
<xs:element name="name" minOccurs="int" maxOccurs="int" />
     <xs:simpleType>
          <xs:restriction base="type">
          …
          </xs:restriction>
     </xs:simpleType>
</xs:element>
```

Declaring Attributes

There are two patterns for declaring attributes:

- With a named type, specified by the type attribute, for example:

```
xs:attribute name="name" type="simple-type" use="how-used" default/fixed="val
```

- With an unnamed simple type, for example:

```
<xs:attribute name="name" use="how-used" default/fixed="value">
     <xs:simpleType>
          <xs:restriction base="simple-type">
          ...
          </xs:restriction>
     </xs:simpleType>
</xs:attribute>
```

The `use` attribute has three possible values: `required`, `optional` or `prohibited`. If there is a `default` or a `fixed` attribute, there must be no `use` attribute. The value of a `default` or `fixed` attribute is a simple type value.

In content model (i.e. complex type) definitions, attribute declarations are placed at the end of all the element declarations [6].

Defining Complex Types

There are four patterns for defining complex types [6].

- Complex type that is not derived from another user-defined type or from a simple type.
- Complex type that is derived from another user-defined complex type by extension.
- Complex type that is derived from another user-defined complex type by restriction.
- Complex type that is derived from a simple type by adding attributes.

The definitions can be global (with a name attribute) or embedded (without a name attribute).

2.2 Related Work

2.2.1 LDAP and XML: Directory Services Markup Language (DSML)

The Organization for the Advancement of Structured Information Standards (OASIS) has approved the Directory Services Markup Language (DSML) as a standard. DSML defines a means of representing directory structural information as an XML document. The intention is that eventually DSML will offer a standardized method of accessing and manipulating data in a LDAP directory.

A DSML document describes either directory entries, a directory schema or both.
Each directory entry has a universally unique name called its distinguished name. A directory entry has a number of property-value pairs called directory attributes. Every directory entry is a member of a number of object classes. An entry's object classes constrain the directory attributes the entry may take. Such constraints are described in a directory schema that may be included in the same DSML document or may be in a separate document [9].

The design approach of the initial version of DSML (DSMLv1) *"is not to abstract the capabilities of LDAP directories as they exist today but instead to faithfully represent LDAP directories in XML"* [9].

Therefore DSML attempts to use XML to map LDAP requests and responses, without creating a more generic XML model that could be used by multiple protocols. It uses XML as a means to transport messages over a variety of protocols, including HTTP and SMTP.

2.2.2 SQL and XML: SQLX (INCITS)

The work done by the SQLX group to combine the use of SQL and XML has been incorporated into the work done by the International Committee for Information Technology Standards (INCITS) Technical Committee H2-Database group. The main aim of this group is to define a standard to develop a well-defined relationship between SQL and XML.

The objectives of the group is to define a standard to cover specifications addressing the following issues [10]:

- Representation of SQL in XML form, and vice versa
- Mapping SQL schema to and from XML schema
- Representation of SQL Schemas in XML
- Representation of SQL actions (insert, update, delete)
- Messaging for XML when used with SQL
- The manner in which SQL language can be used with XML

The mapping of SQL values to XML values is largely determined by the mapping from the SQL data type to the XML Schema data type. XML Schema Part 2 defines simple data types for XML and lexical representations for the values of these types. SQL/XML provides a mapping for each of SQL's scalar data types to an XML Schema data type. The approach SQL/XML has taken is to select the closest possible XML Schema data type for each SQL data type [11].

As is the case for DSML, the SQLX group is concerned only with representing a single language (application), i.e. SQL, in XML format. It is not concerned with creating a more generic XML model that could be used by multiple protocols.

2.2.3 Common Information Model (CIM)

The Common Information Model (CIM) is an object-oriented information model standardized within the Distributed Management Task Force (DMTF) for the purposes of providing a conceptual framework within which any management data may be modelled [12]. It comprises:

- A meta-schema to formally describe the model.
- Core schemas to capture notions that are applicable to all areas of management.
- Standard schema to model vendor-independent information within a small number of specific areas of management.
- Extension schemas to represent vendor specific extensions of standard schemas.

Currently CIM schema (class and instances) is expressed textually in Managed Object Format. XML allows for interchange of this CIM information using XML tools such as XML parsers [12].

This research aspect is focused on a conceptual model, namely CIM. It does not look at using XML as a messaging mechanism across multiple client-server applications.

2.2.4 Simple Object Access Protocol (SOAP)

The use of Simple Object Access Protocol (SOAP) messages over HTTP was considered. SOAP defines standards for XML messaging and the mapping of data types so that applications adhering to these standards can communicate with each other.

The SOAP 1.1 specification, available from http://www.w3.org/, defines a framework for the exchange of XML documents. It specifies, among other things, what is required and optional in a SOAP message and how data can be encoded and transmitted.

The SOAP specification defines envelope structure, encoding rules, and a convention for representing remote procedure calls and responses. These calls and responses are transmitted as SOAP messages over HTTP.

SOAP works by "wrapping" the client and server applications with a SOAP client and SOAP server respectively. It uses Remote Procedure Calls as the method of communication.

The two main types of SOAP messages are those with attachments and those without attachments.

Messages with No Attachments

The following outline shows the very high level structure of a SOAP message with no attachments. Except for the SOAP header, all the parts listed are required.

1. SOAP message

    A. SOAP part

        1. SOAP envelope

a. SOAP header (optional)

b. SOAP body

Messages with Attachments

A SOAP message may include one or more attachment parts in addition to the SOAP part. The SOAP part may contain only XML content. Any part of the message content that is not in XML format, must occur in an attachment part. Note, since an attachment part can contain any kind of content, this means it can contain data in XML format as well.

SOAP messages are sent and received over a connection. The connection can go directly to a particular destination or to a messaging provider. A messaging provider is a service that handles the transmission and routing of messages and provides additional features not available when you use a connection that goes directly to its ultimate destination [7].

Web Services

SOAP messages typically work in conjunction with the Web Service Description Language (WSDL). A Web service can make itself available to potential clients by describing itself in a WSDL document. A WSDL description is an XML document that gives all the pertinent information about a Web service, including its name, the operations that can be called on it, the parameters for those operations, and the location of where to send requests. A consumer (Web client) can use the WSDL document to discover what the service offers and how to access it. In a typical Web services scenario, a business application sends a request to a service at a given URL using the SOAP protocol over HTTP. The service receives the request, processes it, and returns a response [7].

Figure 6 provides a brief overview of the various components used in offering Web Services and describes their interactions. Two services, namely Service A and Service B register their service with a business registry. Each service also defines the WSDL that they will use to communicate with other applications. If Service A wants to communicate with Service B, It locates Service B via the UDDI business registry that service B has registered with. It then sends a WSDL type message enclosed in a SOAP envelope to Service B.

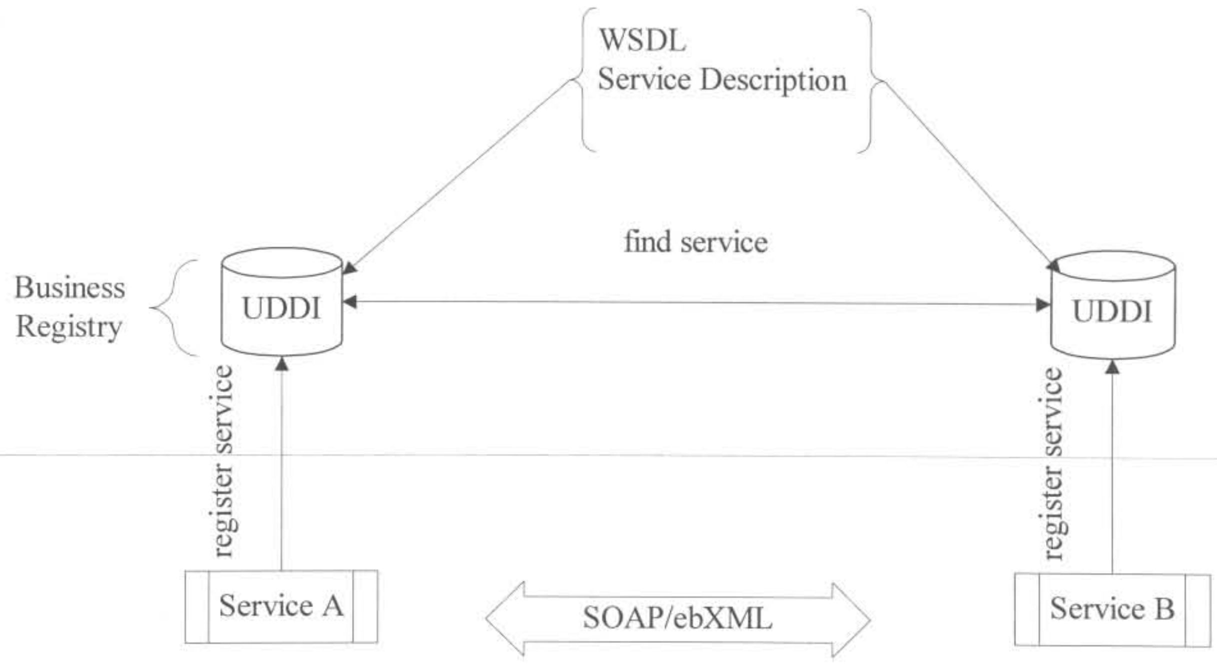


Figure 6: Overview of Web Services

A Universal Description, Discovery and Integration (UDDI) service is a business registry and repository from which you can get information about businesses that have registered with the registry service. A *registry provider* is an implementation of a business registry that conforms to a specification for XML registries.

An XML *registry* is an infrastructure that enables the building, deployment, and discovery of Web services. It is a neutral third party that facilitates dynamic and loosely coupled business-to-business (B2B) interactions. A registry is available to organizations as a shared resource, often in the form of a Web-based service. Currently there are a variety of specifications for XML registries [7]. These include

- The ebXML Registry and Repository standard, which is sponsored by the Organization for the Advancement of Structured Information Standards (OASIS) and the United Nations Centre for the Facilitation of Procedures and Practices in Administration, Commerce and Transport (U.N./CEFACT).
- The Universal Description, Discovery, and Integration (UDDI) project that is being developed by a vendor consortium.

After investigation it was decided not to use SOAP for the following reasons:

- The additional overhead of using a SOAP server and a SOAP client.
- The SOAP specification (version 1.2) is still a working draft and is not yet stable.
- This research is not focused on Web services. XML is a data description language developed and standardised independently of Web services. XML is currently being used for Web services as well as other applications that are not related to Web services.

2.3 Related Work in published Articles.

Studies on the problem of restructuring and reformatting of data as it passes from one software tool or process to another predates the widespread use of the Internet that started in the mid 1990s. Blattner et al. [18,19] in a study on generic message translation, attempted to solve the problem by providing a visual interface that can create a mapping between fields in different message types that specifies which fields have similar semantic content. The Blattner et al. papers were published before the introduction of XML into the computing landscape. However, in their paper, the authors conclude that some sort of "parser-generator" must be constructed to take descriptions for data specifications and create a "translator" between systems.

The following articles while they do not relate directly to the research area are of some interest because the articles demonstrate the increasing use of XML as an interface language between diverse applications.

A study to use XML as the wrapper interface in migrating legacy applications to the browser based Internet platform conducted by Bi, Hull and Nicholl [15], focused on understanding the functionality of the legacy system and the user interaction in the legacy system. Bi et al. developed a thin web based client to interact with the legacy system that was wrapped within an XML application. The focus of the paper was legacy applications but it demonstrated the effectiveness of using XML to interact with multiple legacy applications.

An article by K.L. Law [16] describes an attempt to use XML and LDAP messages to describe network database schema, with the intention that modification of database information could be achieved without taking down the whole system. The authors reached the conclusion that XML's extensibility results in increased flexibility in setting up data content according to different applications and/or different vendors. This conforms to the design of this XML document that if the main elements are defined in a common schema, the sub-elements within the schema can be extended according to the specific needs of the application.

The use of XML to provide a bridge between older network applications and the web browser based Internet platform in a TMN environment is proposed by Lewis and Mouritzsen [17]. While the article focuses on the potential role XML can play in the evolution of TMN, it is interesting to note the increasing number of areas where XML is being considered as a solution in providing a common interface between traditional client-server applications and the Internet.

An article by Peinl and Mitchang [20], which investigates transforming independent, autonomous data sources into a common XML format in order to provide an integrated communication platform for mobile applications.

However, use of XML for generic messaging does not come without some disadvantages, namely slower processing speed caused by the additional overhead of using XML to transform and parse messages. Boedjang et al. [21] in their study of distributed data structures conclude that the performance measurements of applications that run application-specific code are faster then those that use generic message passing software.

Chapter 3 : SYSTEM OVERVIEW

This section provides an overview of the larger project in which this research area is a part of. It attempts to provide the reader with a clearer picture of the domain in which the design was developed and implemented.

3.1 Existing System

The IGUANA gateway is discussed in Chapter 2. The gateway has multiple server applications running on it. The server applications that were the focus of this project are openLDAP server, MySQL database server and ESD server.

The ESD server controls access to all the FAN daemons that provide access to each field area network. The current implementations of openLDAP and the SQL database have been adapted to query ESD to obtain information about a FAN and to update their directories or database respectively.

Each server has a separate client application that requests information about a particular field area network. The gateway will use some sort of security mechanism, either Secure Sockets Layer (SSL) or Transport Layer Security (TLS) to provide a secure connection for transporting messages between client and server.

The client applications will either use data existing on the gateway, for example, openLDAP will use data point information stored in a directory or it may send a query to ESD to obtain the information it requires.

The openLDAP and SQL implementations do not provide all the functionality as provided by ESD, such as REFRESHNODELIST, UPDATENODELIST etcetera. These commands can only be sent to the ESD server directly from the ESD client. ESD itself may use the event and log files on the gateway to respond to certain types of commands.

The figure below provides an overview of the components in the current system and their interactions.

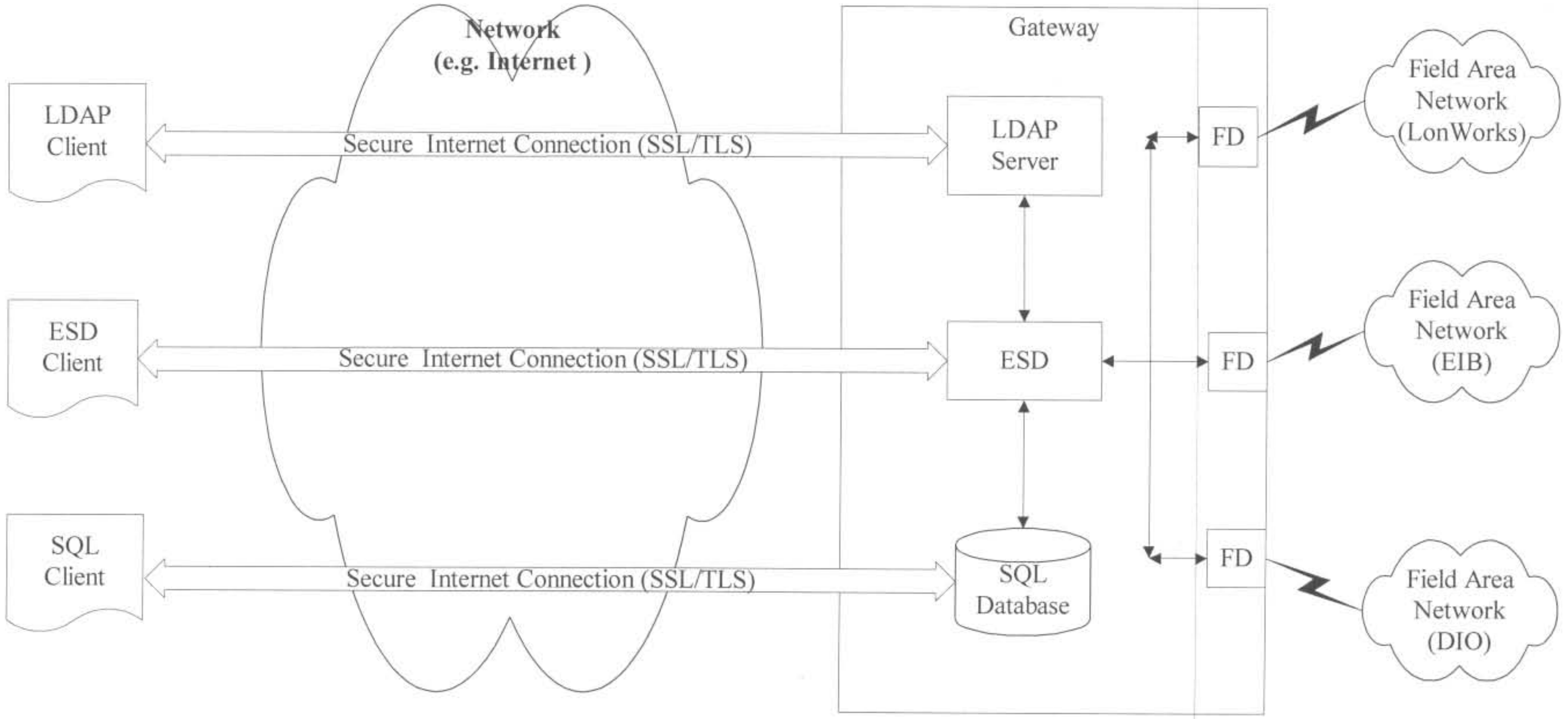


Figure 7: Current System Overview

Note: FD stands for Field Area Network (FAN) Daemon.

3.2 Proposed System

In the common XML messaging system, there is a common client interface to heterogeneous client-server applications that use network transport protocols such as TCP. The server specific messaging protocol is abstracted away from the client side, i.e. the client is provided with a generic user interface to the gateway irrespective of the protocol specific data access method on each server.

The client sends command messages to a common server (the XmlGateway), using a predefined XML format. The XmlGateway module uses the location of the information as a mechanism to determine which server on the gateway the message is intended for. The message is then reformatted into the specific server's message protocol format and sent to the server.

The response from the server is reformatted by the XmlGateway into an XML message response format as specified in a predefined XML document and sent to the client. All clients, therefore only have to process XML type requests and responses. This reduces the complexity of the client application and creates a truer thin client – fat server architecture.

Because messages are sent as an XML document, they can be transported over a number of transport protocols, such as HTTP, TCP/IP and encapsulated within a SOAP envelope. Therefore, if in the future, the gateway was placed behind a firewall, and the firewall restricted access to the HTTP port, the XML messages can still be transported via the firewall.

The XmlGateway currently provides access to LDAP, ESD servers and a SQL database. It is easily expandable and a different server application can be added with minimum overhead.

The proposed system using a common XML messaging protocol is shown in the figure below.

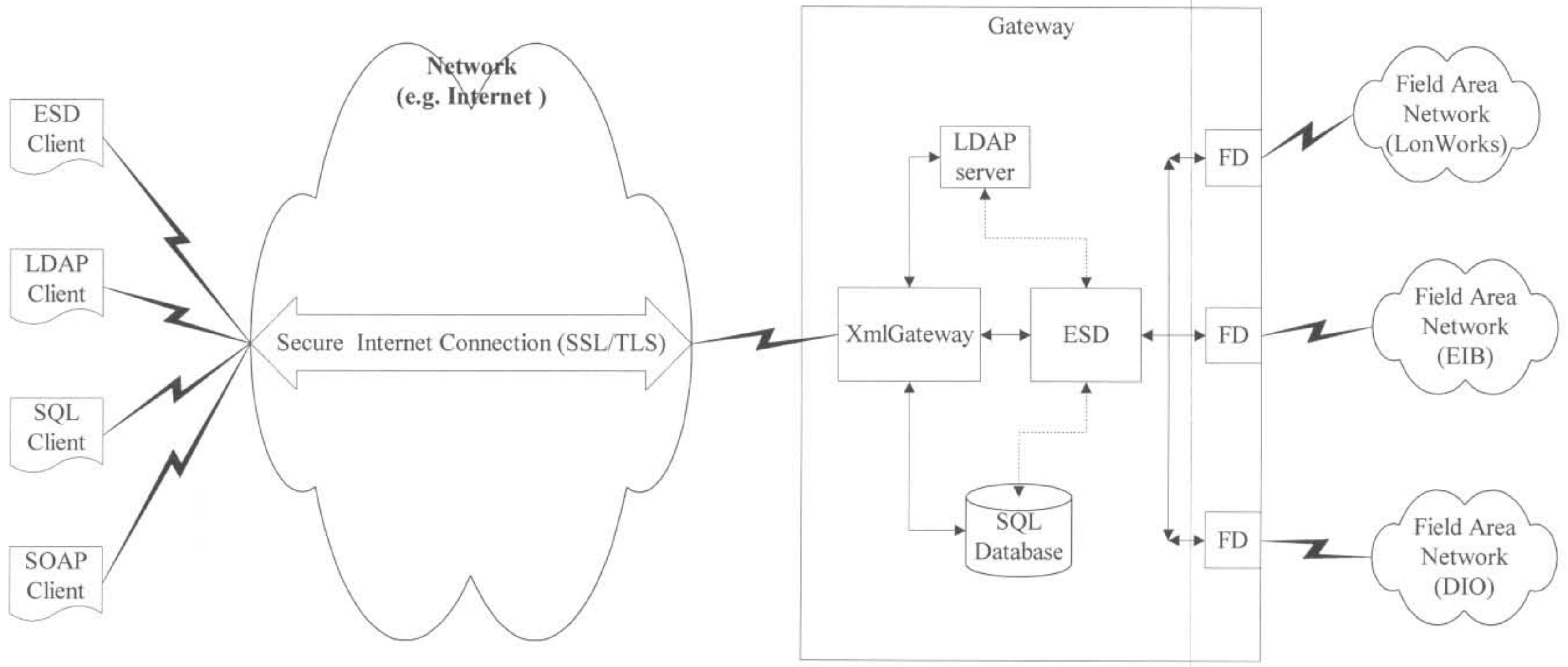


Figure 8: Overview of common messaging system using XML

Note: FD stands for Field Area Network (FAN) Daemon.

The slashed lines between LDAP, SQL and ESD indicate that the system can easily be integrated to the openLDAP and SQL implementations shown in the previous section, which do connect to ESD to obtain FAN data.

Chapter 4 : SYSTEM SPECIFICATIONS

This section describes the system requirements and functional specifications, the software and methodology used and the testing that was done to verify and validate the system.

In addition, a brief overview of the external software packages utilised in the implementation are provided.

4.1 Introduction

The IGUANA gateway currently provides access to data in a field area network node using ESD. It also runs versions of an LDAP server (openLDAP) and a database server that provides access to the same data.

The focus of this research to develop a system to send and receive messages to each of the above-mentioned applications using XML to represent the messages in a common format.

The following sections describe the system requirements identified, the system constraints, and the functional specification.

4.2 System Requirements

The following system requirements were identified:

1. Design and develop a component that allows clients to retrieve data from the IGUANA gateway using XML as the data description language
2. The system should provide access to ESD, LDAP and SQL
3. The component should interface with the above-mentioned applications to access data point and node information, event tables and logs.
4. The system should have some sort of web-based interface to provide online access.
5. An analysis of the performance (speed) for each of the above-mentioned applications should be carried out.

6. Prefer usage of Java as the programming language because it can be used across multiple platforms and does not require re-compilation on different operating systems
7. Prefer use of open source software to limit costs

4.3 System Constraints

The following system constraints were identified:

1. Use XML as the data description language.
2. Application must run on the Linux operating system.
3. There are limited resources available in terms of processor power and memory (486 processor and 8M RAM), but it can be upgradeable if required.

4.4 Assumptions

- The design will not be responsible for security of the gateway data or the security of messages sent and received over the Internet or fieldbus.
- The design will not be responsible for access control and authentication of user login information.
- The XML component may eventually make use of other sub-systems such as SSL or TLS to transmit data securely over the Internet.

4.5 Data Structure

The standard ESD schema is used for compatibility across all server implementations. The reason for this is to enable easier integration between the different application servers in the future and to ensure as close correlation as possible between the different application servers when analyzing the results.

For SQL and LDAP not all commands are implemented in the implementations currently available (i.e. from the IGUANA project). For the sake of completeness to obtain at least one complete comparison, it was decided to add a GENERAL table to the database design that would contain the information (such as eventlistseqno) that is not available in the current table design. Refer to “Addendum A: IGUANA Structured Query Language Daemon (ISQLD)” for a description of the database schema.

It was felt that the database server application would be easier to change and remove the changes from at a later stage if the programs were integrated. However, the updatenodelist and refreshnodelist functionality that is specific to ESD is not implemented. The LDAP implementation only implements the objectclasses specified in the documentation [5]. Refer to “Addendum B: IGUANA LDAP Schema” for a description of the LDAP schema.

4.6 Functional Specification

The following diagram describes the main functional blocks in the project. The arrows indicate the direction of data flow. A description of each functional unit (FU) or user interface (IF) is provided after the diagram.

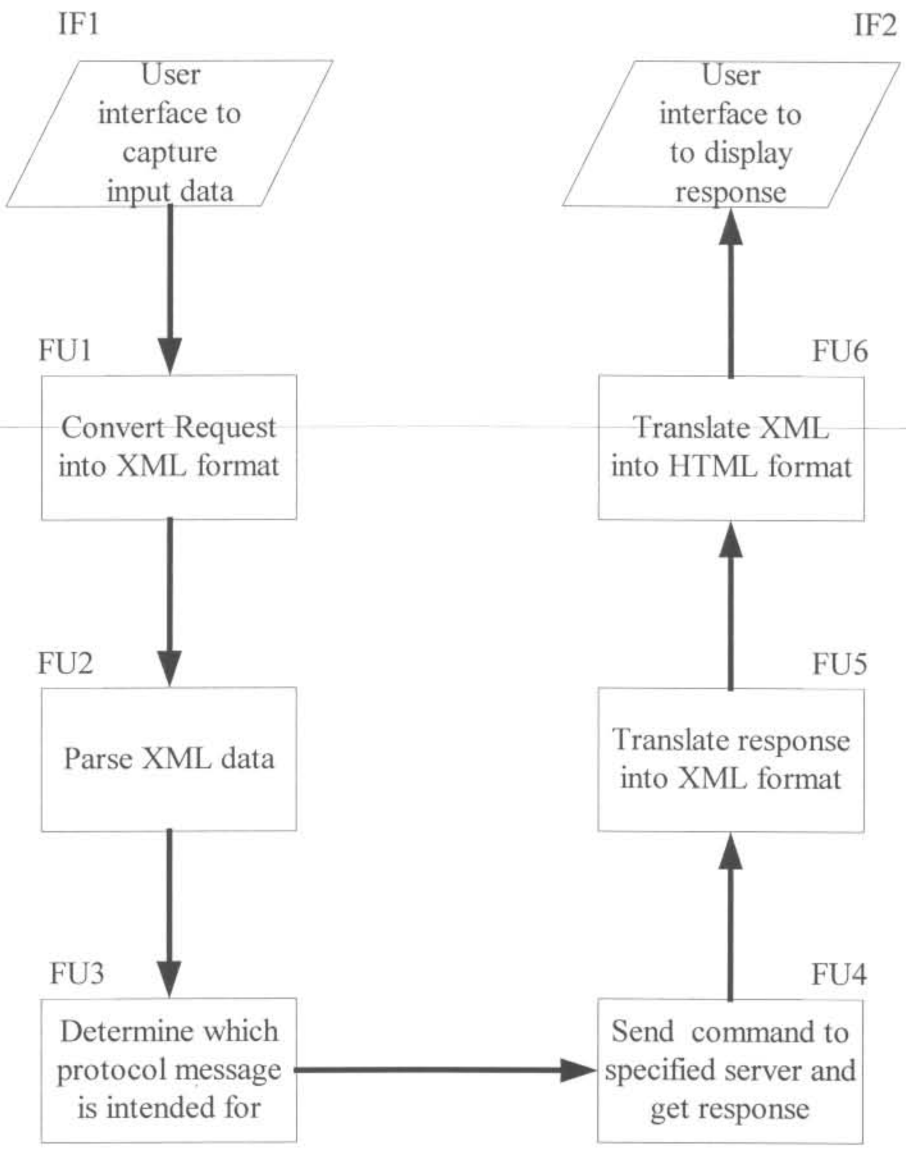

IF1
User interface to capture input data
FU1
Convert Request into XML format
FU2
Parse XML data
FU3
Determine which protocol message is intended for
IF2
User interface to to display response
FU6
Translate XML into HTML format
FU5
Translate response into XML format
FU4
Send command to specified server and get response

Figure 9: Functional block diagram of the proposed system

IF1: This is a web-browser type user interface. The screen displays forms to capture user input. There are separate forms for each command. It comprises Java Server Pages (JSPs) that display the forms used to capture user input per command.

FU1: This functional unit converts the HTTP request into the predefined XML document format.

FU2: This functional unit parses the XML data stream it receives and stores the information as element-value pairs.

FU3: This functional unit uses the location of the data to determine which application server the request should be sent to.

FU4: This functional unit connects to the specified application server and sends the request command to the server. It waits for a response from the server.

FU5: This functional unit converts the response received from the application server into the XML format as defined in the XML Schema.

FU6: This functional unit translates the XML response data stream into HTML format as specified by a selected stylesheet.

IF2: This user interface displays the response to the request sent earlier in HTML format.

The main design components are FU2, FU3, FU4 and FU5. They are concerned with processing an input XML data stream, sending a command to the correct application and processing the command response back into an XML data stream.

The units: FU1 and FU6 are peripheral and can easily be replaced if a client applications decides to send an XML data stream directly to FU2 and to receive an XML data stream back as a response.

The IF1 and IF2 and FU1 and FU6 blocks provide completeness in the system to show how it can be used with a browser based interface.

4.7 Software Methodology

The methodology followed was an iterative software development lifecycle methodology using a feedback based waterfall method. This means that after completing a needs analysis by determining the system requirements and system constraints, an initial system architecture and high-level design was developed. This architecture and design was then discussed with the relevant stakeholders and where applicable, the design was reworked.

The design was then implemented for the ESD application only using the Java programming language on the Linux (Red Hat) operating system. The design and XML model was again refined as implementation progressed. Each section of implementation was initially developed as independent entities and tested individually. The final system was integrated to provide a complete application that consisted of a user interface and backend processes working together.

The system was shown to relevant stakeholders and changes made. Because the system is designed to enable new client-server applications to be added on with minimum additional coding time required, the MySQL and openLDAP implementation were then developed and tested.

4.8 Testing

The testing of the system was carried out on a micro and macro level. Initially, a simple test program was built to ensure that a connection to the server application could be established and messages processed.

Then, separate test programs were developed for each server implementation which verified that XML and non-XML messages are correctly processed when sent to the specified server and that the respective XML or non-XML response messages were received.

For the front-end system, a proof of concept test program was written to test if an XSLT program would correctly transform XML into HTML format.

Finally the entire system was integrated and the actual application was tested from user input to backend server applications, through to the conversion of responses into HTML format.

4.9 External Software Components

The following section lists the external software applications that were used and provides a brief description of each application.

The software components used are all open source applications. Open source means that it is possible for anyone to use and modify the code, as it is freely available (generally for non-commercial purposes).

4.9.1 Database

The database used is MySQL. MySQL is a multi-user, multi-threaded, relational database management system. Clients may connect to the MySQL server using sockets or named pipes. There also are ODBC and JDBC drivers available that allow application programs to connect to MySQL (refer to www.mysql.com for more information). The database was chosen for the following reasons:

- It is an open source application.
- It is a fast, reliable, easy to use relational database system.
- It supports ANSI SQL.

4.9.2 Directory Server

The directory server used is OpenLDAP. The OpenLDAP program was originally developed as a project at the University of Michigan. Further and future development is now handled by the OpenLDAP foundation. Refer to www.openldap.org for more information and specifically to references [24, 25, 27]. The OpenLDAP application was chosen for the following reasons:

- It is an open source application.
- The OpenLDAP implementation supports the complete LDAP functionality needed for setting up an LDAP service on a Linux machine.
- It conforms to the LDAP standards.
- It is the LDAP service currently used in the IGUANA project.

4.9.3 Web Server

The web server user is Tomcat (version 4). Tomcat is a Java Servlet and JSP container developed by The Apache Software Foundation (www.apache.org). Tomcat was chosen for the following reasons:

- It is an open source application.
- It is relatively easy to configure and use.
- It is reliable and stable and supports usage of JSP.
- It is part of the well known and widely used and supported Apache suite of enterprise products.

4.9.4 XML Parser

The XML parser used is Xerces. Xerces is a reliable and easy to use tool for XML parsing and generation. Xerces was chosen for the following reasons:

- It is an open source application.
- It is relatively easy to configure and use.
- It is available for both Java and C++.
- It implements the W3C XML and DOM (Level 1 and 2) standards, as well as the de facto SAX (version 2) standard.
- The parsers are highly modular and configurable.
- It provides initial support for XML Schema (draft W3C standard).
- It is part of the well known and widely used and supported Apache suite of enterprise products.

4.9.5 XML Translator

The XML translator used is Xalan. Xalan provides high-performance XSLT stylesheet processing. The reasons for using Xalan are:

- It is an open source application.
- It is relatively easy to configure and use.
- It is available for both Java and C++.
- It implements the W3C XSLT and XPath recommendations.
- It is part of the well known and widely used and supported Apache suite of enterprise products.

4.9.6 Programming Language

The programming language used is Java (version 1.4). The reasons for using Java are:

- It is a relatively open standard programming language.
- It can be used across multiple platforms.
- It supports object-oriented programming.

4.9.7 Operating system

The operating system used is Red Hat Linux. The reasons for using Linux are:

- It is an open source application.
- It is relatively easy to use.
- All the above-mentioned software applications are compiled to work on this operating system.

Chapter 5 : XML MODEL

This section describes the methodology followed in designing the XML model that is used in the implementation. An XML model was developed to represent client server messages from heterogeneous applications in a generic format.

The next sub-sections describe the commonalities between the different server applications, the XML model design and XML document structure. To the best of our knowledge the XML model proposed is novel.

5.1 Identification of similarities between the heterogeneous applications

The following similarities were identified:

- All applications are client-server applications.
- All applications require connection information.
- All applications send requests in text-based format.
- All applications receive responses in text-based format.
- The applications operate on similar request-response format.
- All applications currently require (or will require) authentication information.
- All applications have mechanisms to read a specified data point/field's value.
- All applications have mechanisms to modify a specified data point/field's value.
- All applications have mechanisms to insert new data.
- All applications have mechanisms to delete data.

5.2 XML Document Design

The model was designed by initially focusing on how messages are sent between the client and server.

In client-server architecture, the server behaves as a slave and the client as the master. A typical client-server application using TCP as the message transport mechanism, will function as follows:

1. A server will stay in a listen state, which means that the server application listens for input data on a specific port.
2. The client will send a connection request to the server.
3. The server will validate the client's authentication details.
4. If the client's authentication details are valid, the server will inform the client that the connection is accepted.
5. Otherwise the server will inform the client that the authentication details are invalid and that the connection will not be established.
6. The client can send a message (command) to the server.
7. The server will process the request and return a response to the client.
8. After the client has processed all requests, it informs the server that the connection will be closed.
9. The client then closes the connection.
10. The server remains in the listen state, in case it has other clients that are still connected to it or may want to connect to it.

From the above description, the following commonalities in most client-server applications are identified:

1. The client has to provide connection information to connect to the server. This connection information could be the host name on which the server resides, the port number on which the server is listening on, a timeout value to wait while attempting a connection, the context or directory in which the server is located, the data source name and JDBC/ODBC driver details, etcetera.
2. The client has to provide the server with authentication details, so that the server can verify that the client has access to the information that the server will be able to provide. This is for security reasons so that rogue client applications that may have malicious intent are not allowed to gain access to the information that the server provides. Typical authentication information required is a user name, a user password (or access code) and the role of the user (i.e. some users may have more privileges to information than other users).

3. The messages, sent between the client and the server, are of the request-response type. The message sent from a client is a request, which contains some specific command. The message sent from the server is a response to the specific command.

Therefore, from the above, the following information is required and has to be included in the XML model, namely:

- Connection Information, i.e.

```
<connection-info>
         <connection-url>"   … " </connection-url>
</connection-info>
```

- Authentication Information, i.e.

```
<authentication-info>
         <auth-name>"   … " </ auth-name>
         <auth-code>"   … " </ auth-code>
         <auth-role>"   … " </ auth-role>
</authentication-info>
```

- Request message with a specific command

```
<request>
         <command>"   … " </command>
           … [command parameters] …
</request>
```

- Response message to a specific command

```
<response>
         <command>"   … " </ command>
           … [response details] …
</response>
```

The command value can be the specific command name used in the application. Note, the schema does not specify what the input elements within a command should be. Each command has specific fields that it needs to parse to a server application in order for the server application to correctly process the request.

The advantage of not specifying the elements within a command element in the generic schema is that additional server applications with different commands and command parameters can use the same schema (with the proviso that a unique data identifier is one of the sub-elements included in the command hierarchy). This is because the XML gateway only requires the following elements: the connection-location, the authentication-info, and the request, response, command and unique data location (or protocol) elements.

All other elements are stored as element-value pairs in a hashtable that is sent to the relevant server application. The value of the command element is irrelevant, because the XML gateway application assumes that the server application to which the command value is sent, will be able to correctly interpret and process the command.

Which application to send request to?

The next question that needs to be asked is how to differentiate between the different applications?

The requirement states that there should be a common messaging structure that is independent of individual protocols (application message formats). In addition, the XML schema should allow the client side to be able to use the same user interface to access the same type of data from multiple server applications.

Data and node points and events have an identifier that is unique to a data point, node or event. To solve the problem of deciding which application to send the request to, the location of the data is used, as the determinant in deciding which application server the request should be directed to. Many request commands require some sort of unique address or field that indicates where the data can be located as an input parameter. This address may be a directory location, a database table and or field name, a file name, a network node and file name and many other possible location type variables.

The address content is used in the implementation to determine which server application to send the message to. The first part of the address before the end punctuation point (left side of the address taken as beginning of address) indicates the type of server application. For example if the message address is "`SQL.SYS.GATEWAY@DataHostAddress!gateway`", then the characters before the first punctuation point is "`SQL`". This indicates that the request is intended for the database server specified in the connection URL. A typical XML document using this approach is shown below.

```
uest>
  <command>read
       <data-location>SQL.SYS.GATEWAY@DataHostAddress!gateway </data-locati
       <data-encoding>STRING</data-encoding>
  </command>
quest>
```

This method works well where the command requires input data that can be used to identify the application server the request is intended. However, some commands may not require any input data. To solve this problem, the following three possible solutions were investigated.

The first solution would force the client application to send through a dummy input field that contains a string identifier that can be used to determine the server application. A separate user form would be used for each server application, with a hidden field indicating which server application to send the request to.

The problem with this approach is that, it requires new user forms to be created every time a new server application is added to the gateway. This would create additional overhead. An alternative solution may be to create a form that allows the user to enter in a location value (similar to the address location) that is used solely to differentiate which server application to use. Therefore as additional server applications are added on the server machine, the client side should remain relatively stable with no changes but should still be able to access the new server application data.

The second solution involves having an additional element in the XML Schema that will identify the server application. The problem with this approach is the same as the previous solution, namely that it requires new user forms to be created every time a new server application is added to the gateway. The forms would indicate which server application the request will be routed to. The XML document for this type of solution is shown.

```
<request>
         <protocol>LDAP</protocol>
          <command>nodelist</command>
          <command>eventlist</command>
</request >
```

The third solution involved attempting to connect to each server application specified in the connection URL. The request is sent to the first server application that is available and the result returned to the client. This solution works well if there is only one server application up at the time and it happens to be the server application the client intended the request to be sent to. However, if these exact requirements are not met, then the flaws in this solution become apparent. For example, if more than one server application is available, the XML gateway may send the request to the incorrect server application.

The XML model uses either the additional protocol element or a dummy location identifier on the client side to indicate which server to route the command to. This provides the user with the flexibility of not requiring that all commands have a data location identifier element or that all requests require a protocol element.

Solution two is suitable for batch type interactions where the server identifier can be passed as an input parameter to the batch application when it is run. This negates the need for separate client applications on the client workstation.

Solution one is suitable for user interfaces such as web browsers where a common user interface can be used to send messages to multiple server applications.

Chapter 6 : APPLICATION ARCHITECTURE

This section describes the application architecture of the proposed system. The external interfaces (from a black box point of view) to the application are also identified.

6.1 Application Architecture

Previous client-server architecture consisted of a relatively thick client application connecting to a server data-source type application that provided the client with requested information. With the advent of the WWW and Internet technologies, application architecture has moved to three-tier (or more as in n-tier) architecture, where the application is further separated into three distinct layers. Each layer concentrates on specific areas and tasks, such as presentation, interpretation of business logic or obtaining data.

The client application has gradually had its overhead and complexity reduced and it is increasingly common that thinner client applications which mainly serve as user interfaces to capture and display user input are becoming available. The computation intensive business and data instructions are increasingly being performed on the server machine. The server machine may consist of one or more machines. In web applications the business and data logic are typically located on separate machines to enable each machines resources to be used optimally for specific tasks.

The application architecture used in this implementation is a typical three-tier architecture. The application architecture consists of the following major components:

1. The presentation layer.
2. The business layer.
3. The data layer

Currently, the business and data instructions are processed on a single server machine, but the data sources can be moved to another server without impacting on the application. In addition, most of the presentation pre-processing is done on the server, so that the client is presented with only HTML type forms that require user input or which displays the server response to a request command. The application architecture is shown in Figure 10.

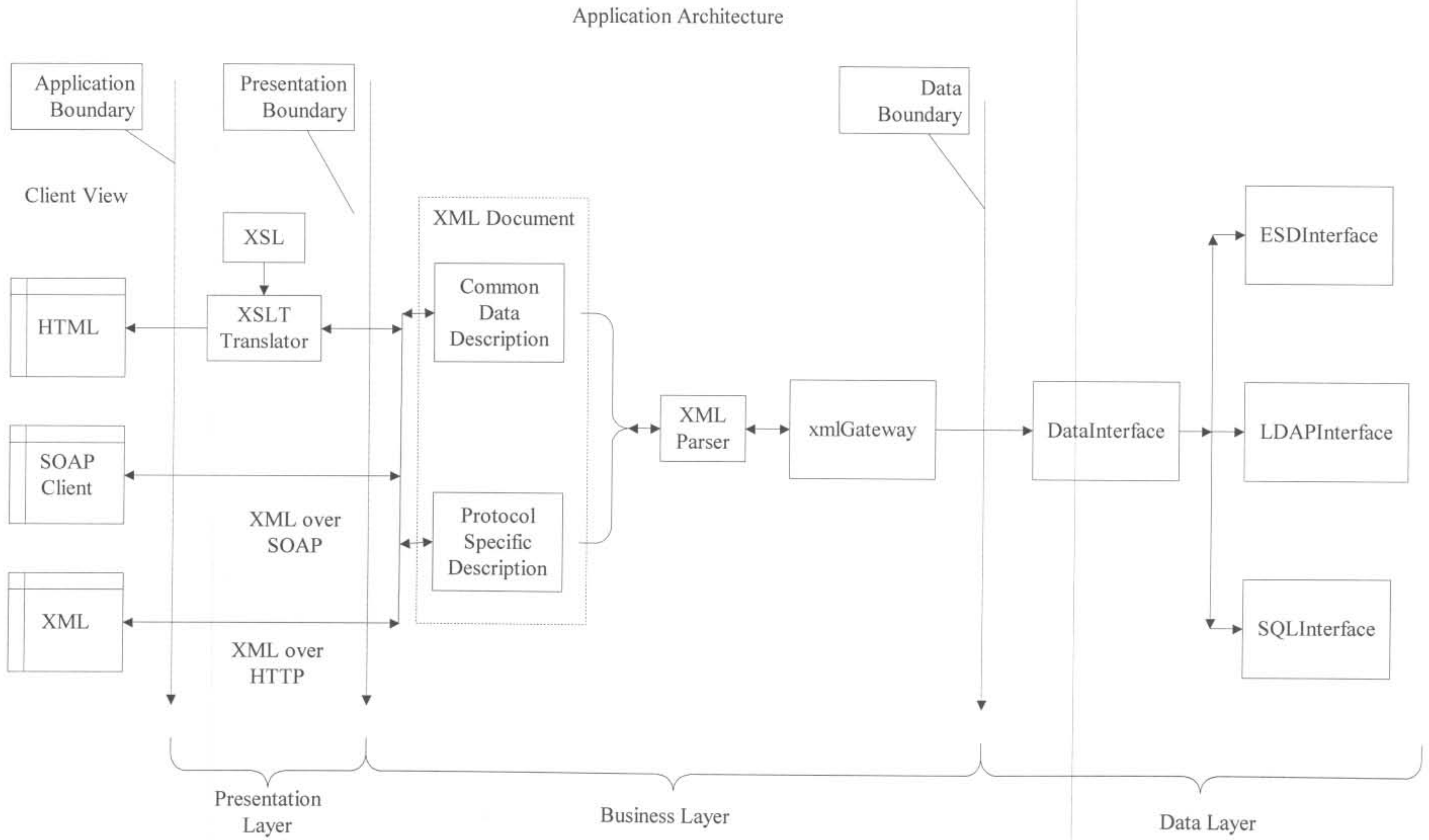


Figure 10: Application Architecture

6.1.1 Presentation Layer

This layer represents the user interface. The user interface consists of web pages in the form of Java Server Pages (JSPs). These pages display forms with input fields and tables containing the server's response.

The XML document can be returned to the client in three possible ways:

1. As an XML document using HTTP as the transport protocol, i.e. the user views the XML syntax in a web browser that has an XML parser such as Internet Explorer.
2. The XML document can be transformed into an HTML form using a stylesheet and the eXtensible Stylesheet Language for Transformations (XSLT)
3. The XML document can be enclosed in a SOAP envelop and sent over HTTP to a SOAP client.

The implementation uses the second option. XSLT is a programming language for transforming XML data. The XSLT processor receives an XML data source and stylesheet and transforms the XML into the format specified by the stylesheet. The result can be XML, HTML or plain text. In this case, XSLT is used to convert XML data to HTML for display in the browser.

The stylesheet is loaded as a stream of characters and parsed into a tree. The XSLT processor applies the stylesheet to the input tree; and renders the result using the stylesheet for HTML.

As Figure 11 shows, it is relatively easy to construct pipelines of XML processors in which each processor receives XML data, does some transformation and/or computation on it and sends the result as XML to the next processor.

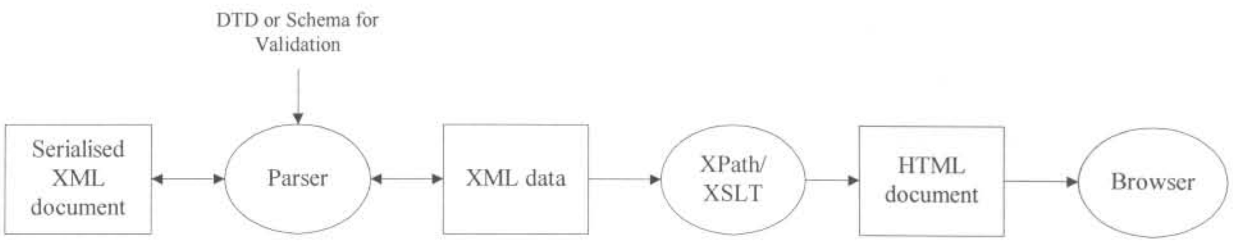


Figure 11: XML processing for browser

6.1.2 Business Layer

This layer interprets the XML data and determines what command to execute, i.e. type of request to be sent to the appropriate data layer for processing. The business layer returns a response to the presentation layer.

The business layer is where the interpretation of the XML data occurs using the standard XML API's. The business layer interprets the XML data and metadata. Depending on the data it determines the type of protocol to use, the type of command to execute on the data and initiates communication with the data layer to perform the command on the specified data value. After it completes processing of the command, it processes the response into the correct XML structure and sends the response to the client layer.

Figure 12 describes the procedure of the business layer.

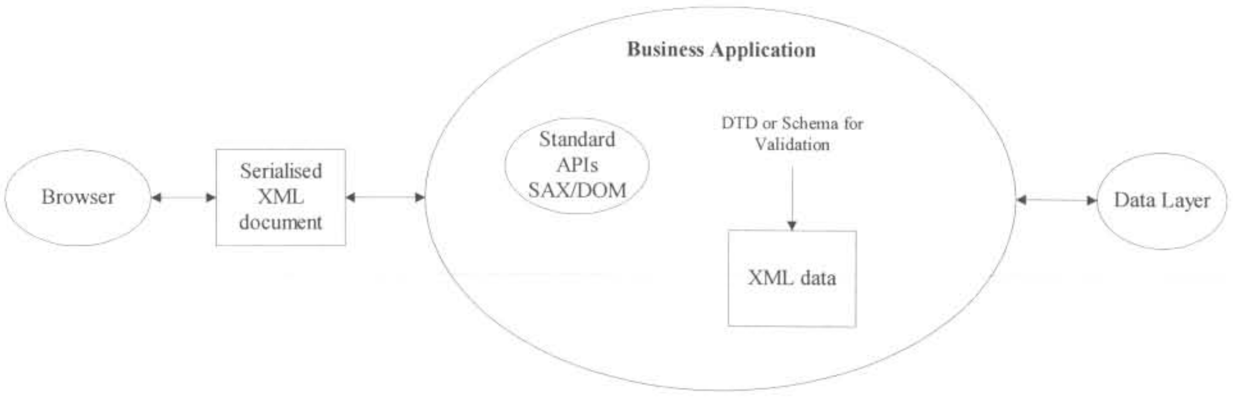


Figure 12: Representation of the business layer

6.1.3 Data Layer

This layer accesses a specified data source and retrieves, updates, adds or deletes data in the data store. Depending on the server application type, the business layer instantiates an instance of a specific data class that will be able to access data from the specified server application.

These are currently three data classes, namely an ESD, LDAP and SQL class. When new server applications are added, supporting data interface classes will be added in this layer.

6.2 Interfaces to External Applications

The scope of the research and the identified external applications it interacts with are identified in Figure 13 below.

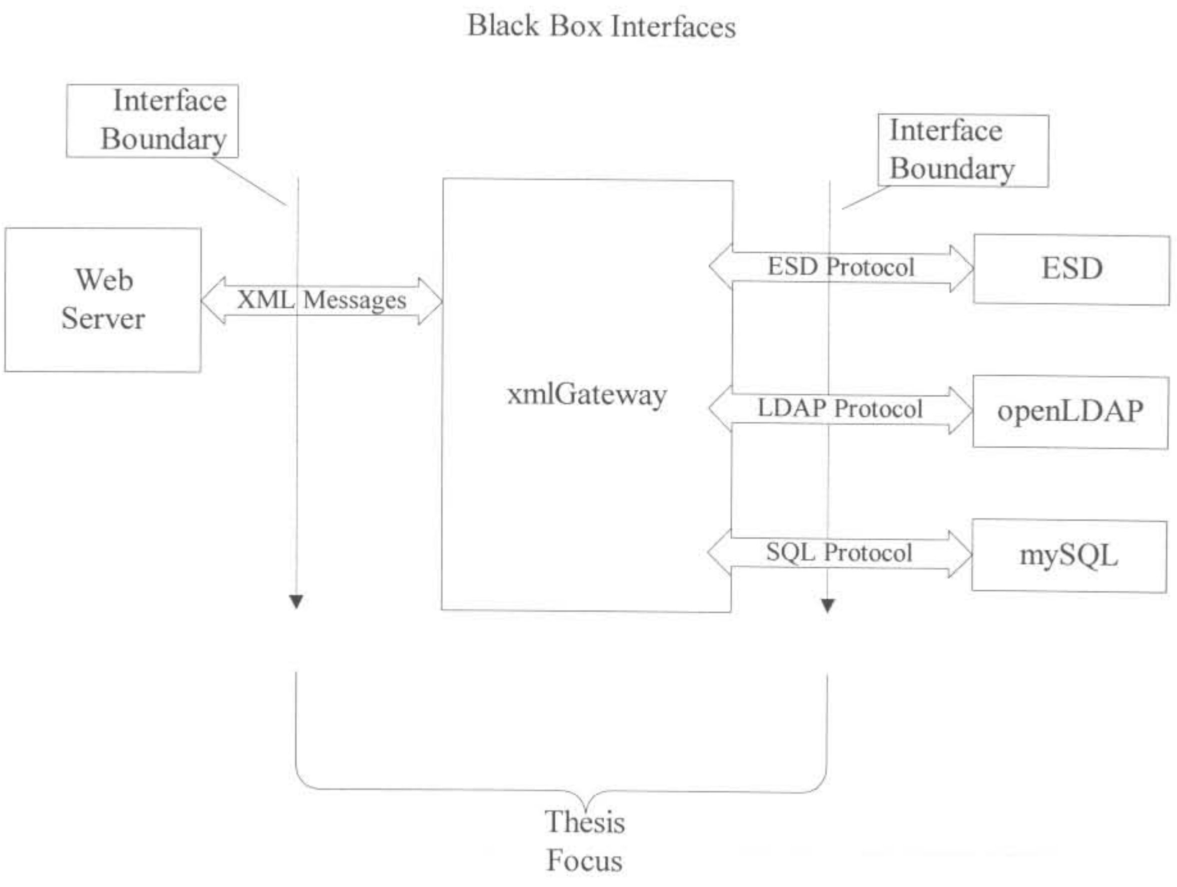


Figure 13: External Interfaces

The interface to the XML gateway on the data side is the ESD, LDAP and SQL servers. Application server specific protocols are used to retrieve information from the servers. The interface to the XML component on the presentation side is the Web server. The Web server receives HTML requests from the client and parses it to the specified Java class for processing.

Chapter 7 : SOFTWARE DESIGN

The following section describes the design of the software components and their interactions. Each software class uses or is used by other classes in the application. The different software classes will be discussed according to their packages. A Java package is a set of classes and interfaces that perform related tasks.

7.1 Introduction

The design document makes extensive use of figures to explain the design. There are three types of figures:

1. The package diagram
2. The class diagram
3. The flowchart

7.1.1 The package diagram

The package diagram identifies the packages that comprise a system and dependencies between packages. Arrows between packages indicate that the classes of one package depend on the classes of another (indicated by the arrow tail). In this document, only the packages developed for the application are shown, i.e. any Java packages used in the implementation are not shown in the diagram.

7.1.2 The class diagram

The class diagram identifies classes, interfaces and their relationships. Arrows between classes indicate that a class (indicated by the arrow tail), extends from another class, implements an interface or uses a class.

7.1.3 The flowchart

The class flowchart describes the flow of logic for a particular command within a class. The arrow tail indicates the logic flow.

7.2 Design

7.2.1 The Presentation layer

The FieldBusBean class is a class that processes the HTTP requests sent by the JSPs. It parses the messages to the XML gateway class and translates the response received from the XML gateway into HTML format.

The following figure shows the package diagram. The fieldbus package uses classes from the gateway package and the library package.

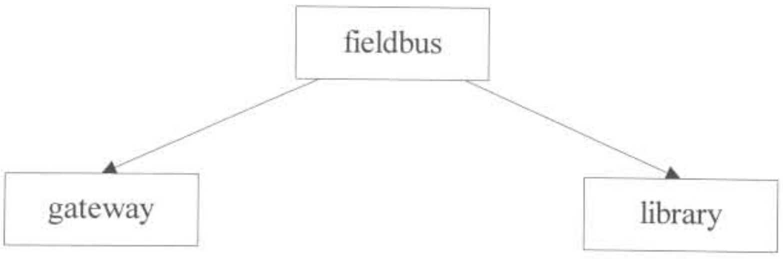


Figure 14: Fieldbus package diagram

The following figure illustrates the class diagram. The FieldBusBean class uses the ParseXmlToProtocolCommands class from the gateway package and the JSPException class from the library package.

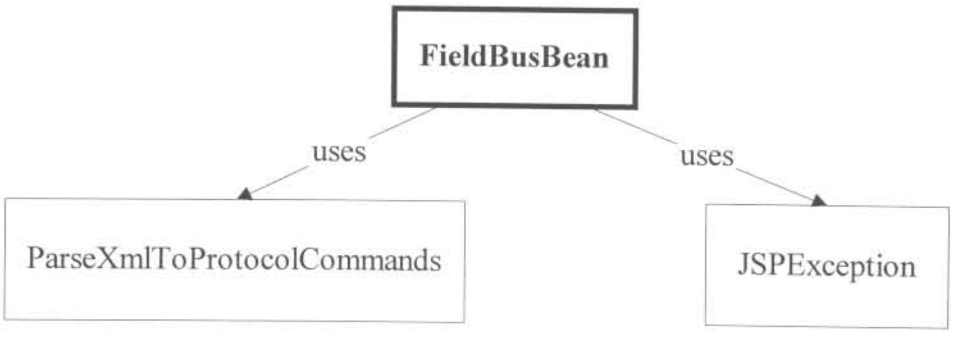


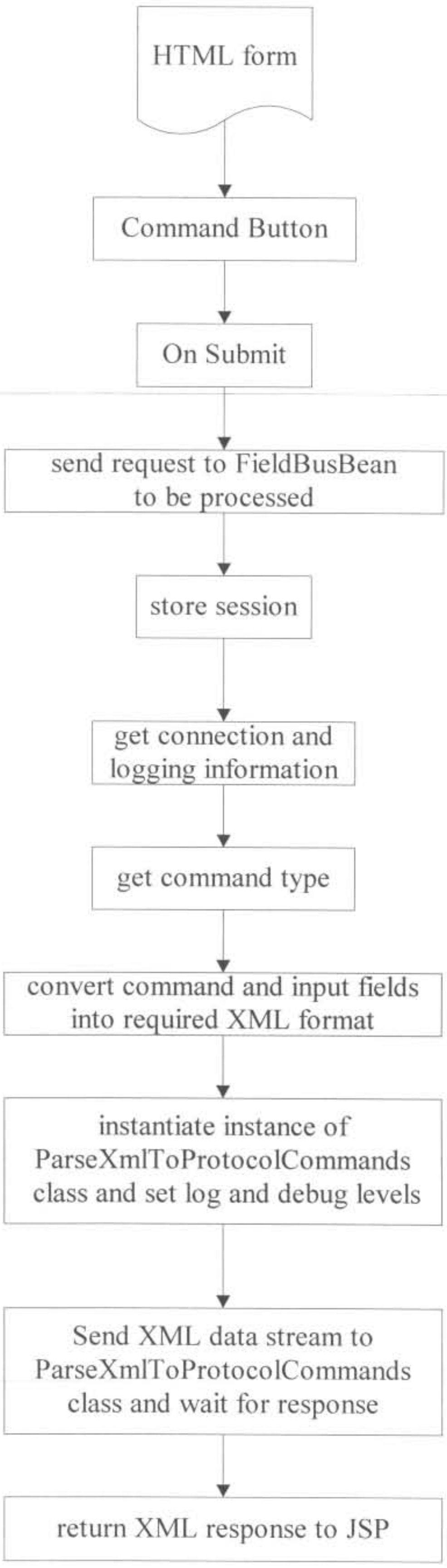
Figure 15: FieldBusBean class diagram

The following sections describe the main functionality of the FieldBusBean class.

Sending a command

The flowchart in Figure 16 (page 67) illustrates the logical flow of data from user input to the server application.

Figure 16: Sending a command from the front end


HTML form
Command Button
On Submit
send request to FieldBusBean to be processed
store session
get connection and logging information
get command type
convert command and input fields into required XML format
instantiate instance of ParseXmlToProtocolCommands class and set log and debug levels
Send XML data stream to ParseXmlToProtocolCommands class and wait for response
return XML response to JSP

Translating a response into HTML

The flowing figure describes the steps in transforming an XML response into HTML format.

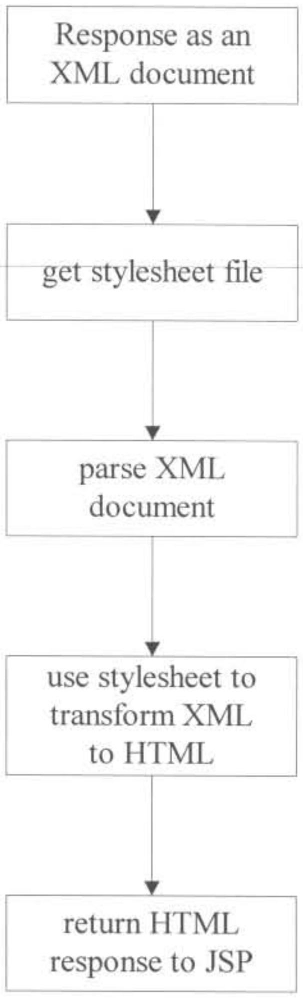
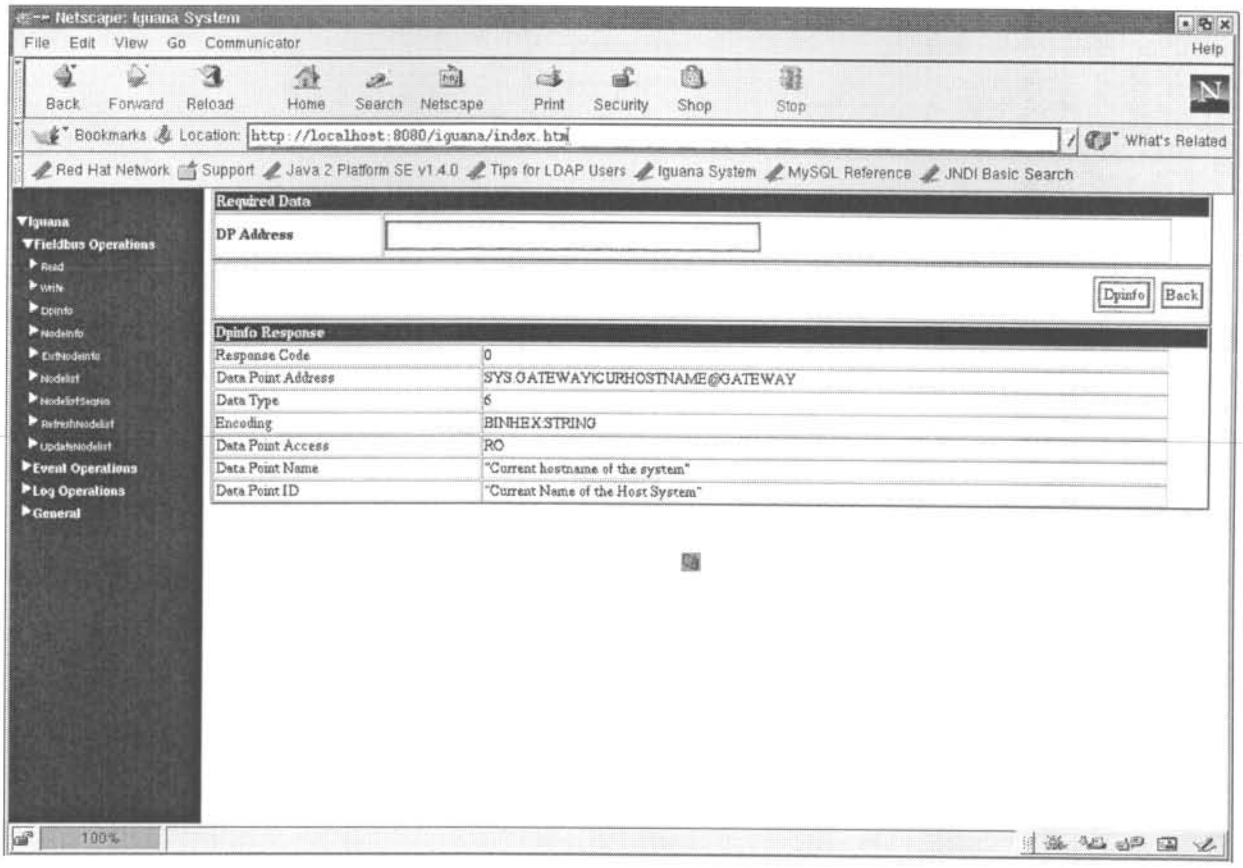

Response as an XML document
get stylesheet file
parse XML document
use stylesheet to transform XML to HTML
return HTML response to JSP

Figure 17: Translating a response into HTML

The snapshot in Figure 18 shows a typical user interface web-screen. The request frame contains an input box, in which the user enters the data point address. The user clicks on the command button ("Dpinfo") to send the request to the application server. The response frame displays the result of the response from the application server. The response from the application server has been formatted from XML into HTML before being displayed.


Netscape: Iguana System
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://localhost:8080/iguana/index.htm What's Related
Red Hat Network Support Java 2 Platform SE v1.4.0 Tips for LDAP Users Iguana System MySQL Reference JNDI Basic Search
Iguana
Fieldbus Operations
Read
Write
Dpinfo
Nodeinfo
ExtNodeinfo
Nodelist
NodelistSeqno
RefreshNodelist
UpdateNodelist
Event Operations
Log Operations
General
Dpinfo
Back
100%

Required Data

DP Address	

Dpinfo Response

Response Code	0
Data Point Address	SYS.GATEWAY.CURHOSTNAME@GATEWAY
Data Type	6
Encoding	BINHEX:STRING
Data Point Access	RO
Data Point Name	"Current hostname of the system"
Data Point ID	"Current Name of the Host System"

Figure 18: Snapshot of user interface for DPINFO command

7.2.2 The business layer

The business layer implements the main functionality of the application. It parses the XML documents and decides which server application to send the request to.

7.2.2.1 gateway package

The following figure shows the package diagram. The gateway package contains the business logic and uses classes from the library, esd, sql, and ldap packages.

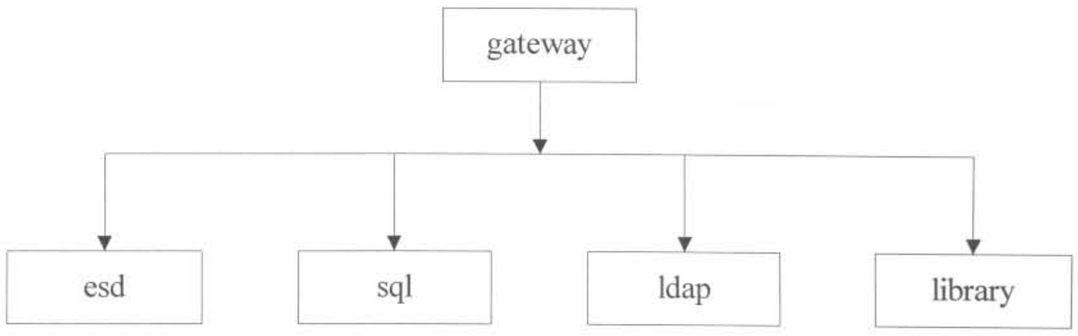


Figure 19: Gateway package diagram

The gateway package consists of the following classes:

- ParseXmlToProtocolCommands
- ProcessProtocolInterface
- ProcessEsdProtocol
- ProcessSqlProtocol
- ProcessLdapProtocol
- ProtocolException

Each of these classes is discussed in the following sections.

Class:: ParseXmlToProtocolCommands

This is the main "brain" of the application. The XML document is parsed and the logic to determine which server application to use, resides in this class.

The self-describing capabilities of XML means that as the XML stream is parsed, specific descriptions are looked for (such as protocol or data location) and the values of these elements are extracted and used to determine the type of application server the message is intended for. The other elements that are command specific are not relevant to the gateway application and are passed to the application specific class as a table of key-value pairs.

The following figure shows the ParseXmlToProtocolCommands class diagram. The ParseXmlToProtocolCommands class uses the LogMessage class (to log warning, error and exception type messages in a log file), the ProcessProtocolInterface interface and the ProtocolException class.

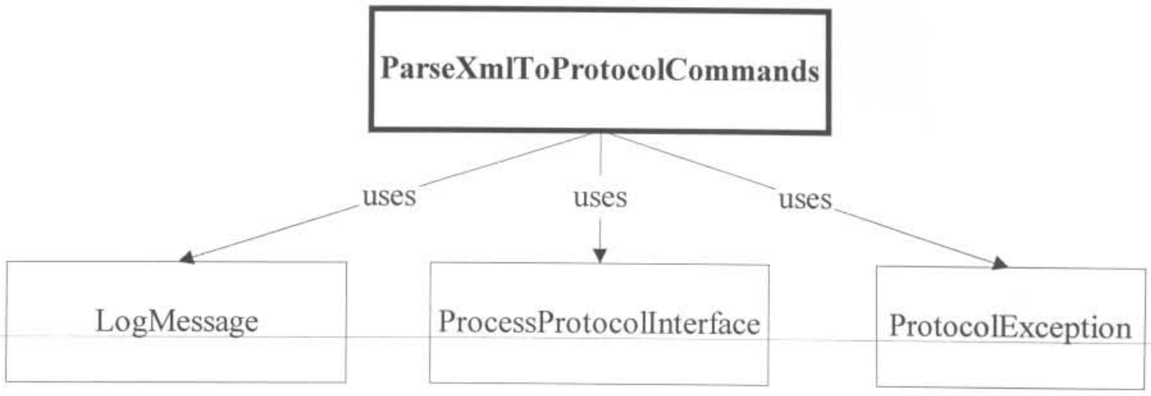
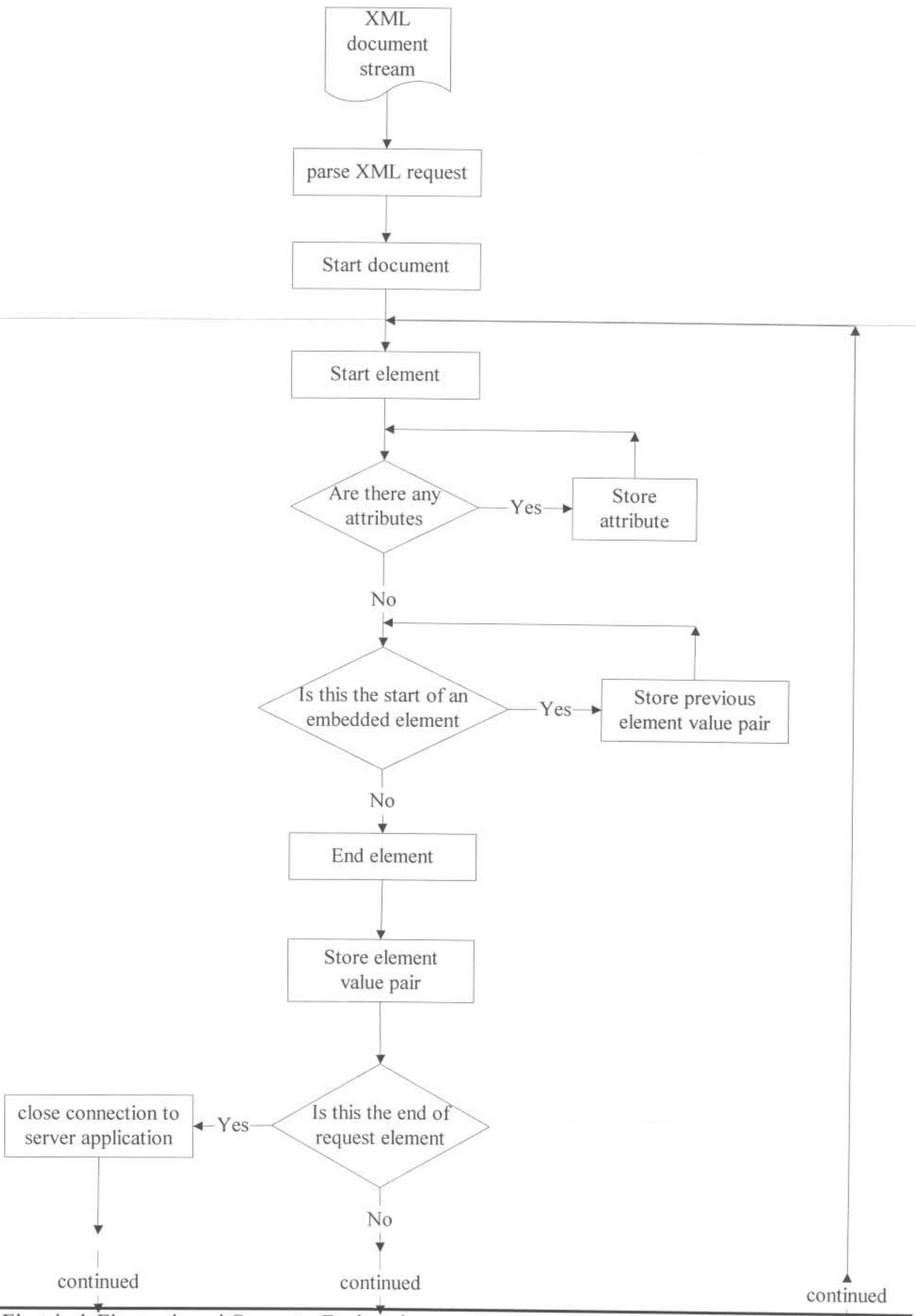


Figure 20: ParseXmlToProtocolCommands class diagram

The following figure illustrates the flow of logic in the ParseXmlToProtocolCommands class.


XML document stream
parse XML request
Start document
Start element
Are there any attributes
Yes
Store attribute
No
Is this the start of an embedded element
Yes
Store previous element value pair
No
End element
Store element value pair
Is this the end of request element
Yes
close connection to server application
No
continued
continued
continued

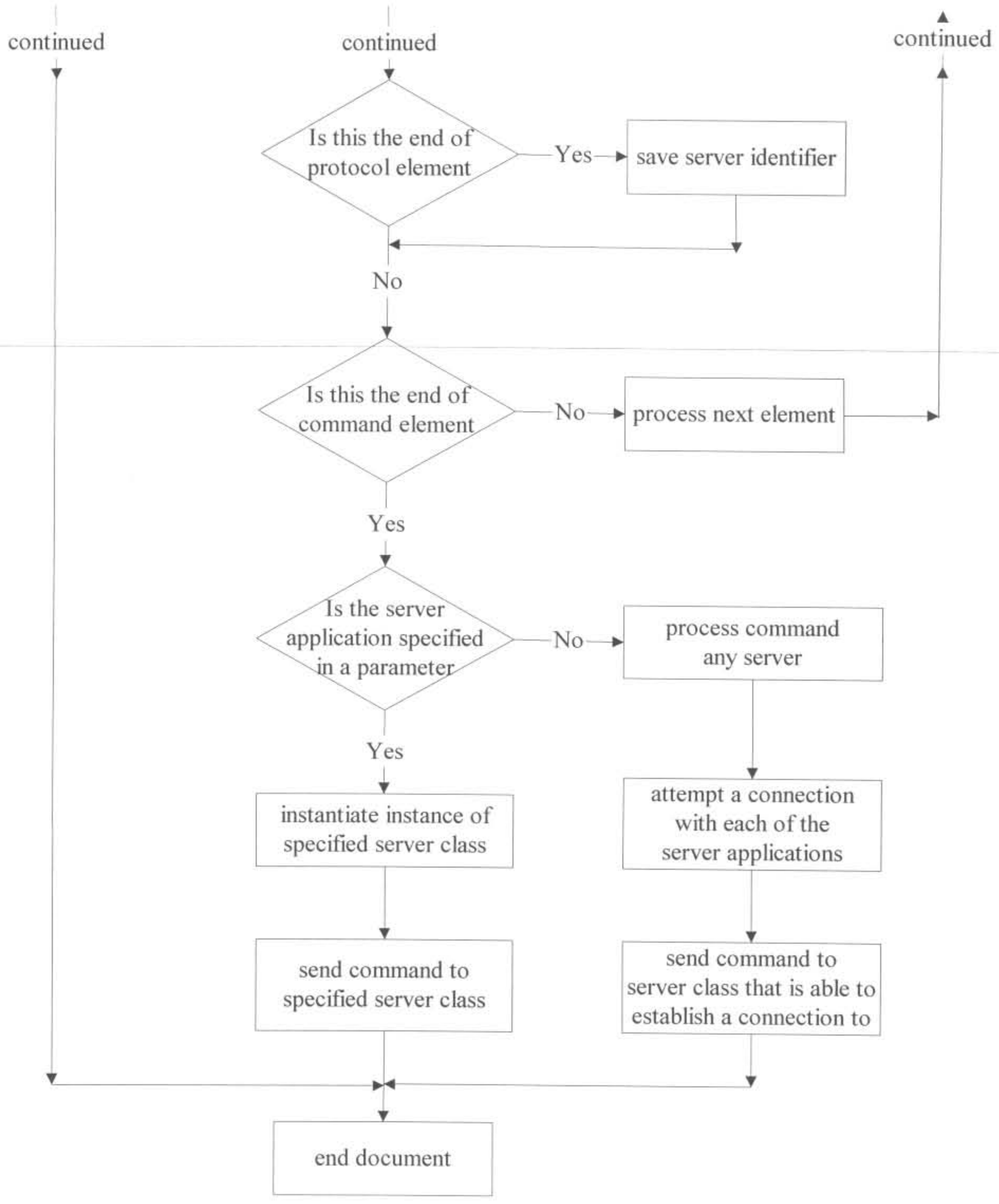

continued
continued
continued
Is this the end of protocol element
Yes
save server identifier
No
Is this the end of command element
No
process next element
Yes
Is the server application specified in a parameter
No
process command any server
Yes
instantiate instance of specified server class
attempt a connection with each of the server applications
send command to specified server class
send command to server class that is able to establish a connection to
end document

Figure 21: Parsing and processing XML data

The class names are stored in a properties file. This properties file is parsed as an input parameter to the ParseXmlToProtocolCommands class on instantiation. A typical example of the contents of the properties file is shown below.

```
esd=za.ac.up.iguana.gateway.ProcessEsdProtocol
sql=za.ac.up.iguana.gateway.ProcessSqlProtocol
ldap=za.ac.up.iguana.gateway.ProcessLdapProtocol
logfile=iguana.log
```

If a new server application is added the code that will identify the application such as sql is added to this properties file together with the class name that needs to be instantiated.

Interface::ProcessProtocolInterface

This interface is provided to allow custom applications to be used by the ParseXmlToProtocolCommands class. As the ParseXmlToProtocolCommands class allows the client application to integrate with multiple disparate types of server applications, it is imperative that it should not have an intimate understanding of the workings of the different server applications. This interface therefore provides this abstraction layer.

Class::ProcessEsdProtocol

This class implements the ProcessProtocolInterface interface. It provides connectivity to the ESD server application. The ProcessEsdProtocol class uses the LogMessage, the EsdCommander and ProtocolException classes. The class diagram for this class is shown in the figure below.

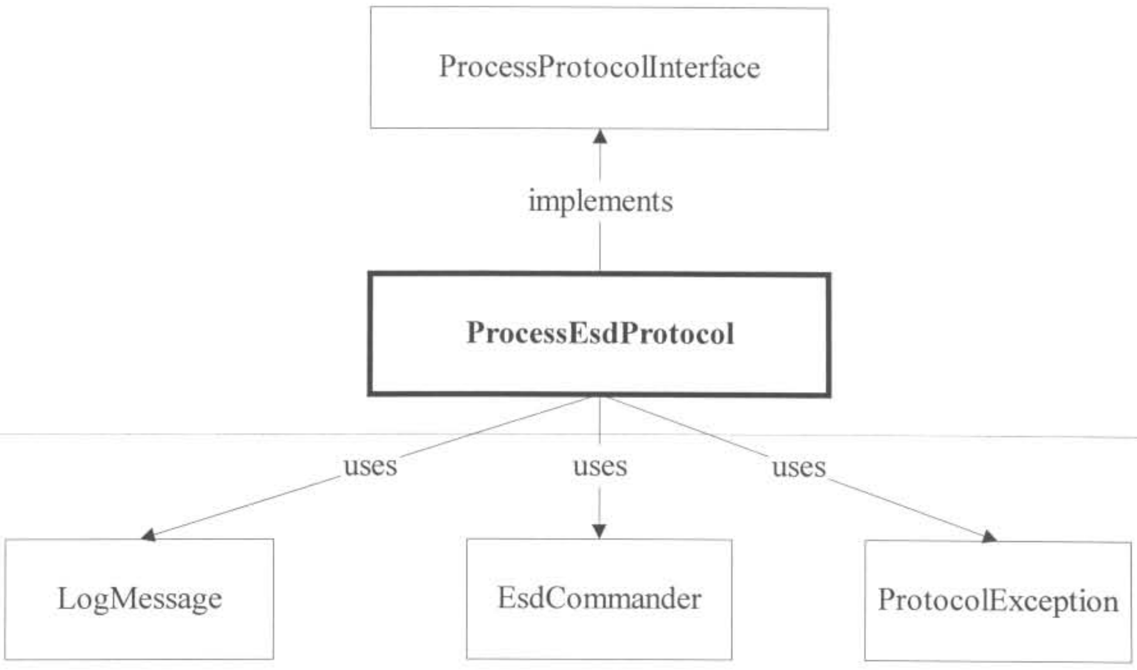


Figure 22: ProcessEsdProtocol class diagram

The main logic flows for this class is shown in the figure below.

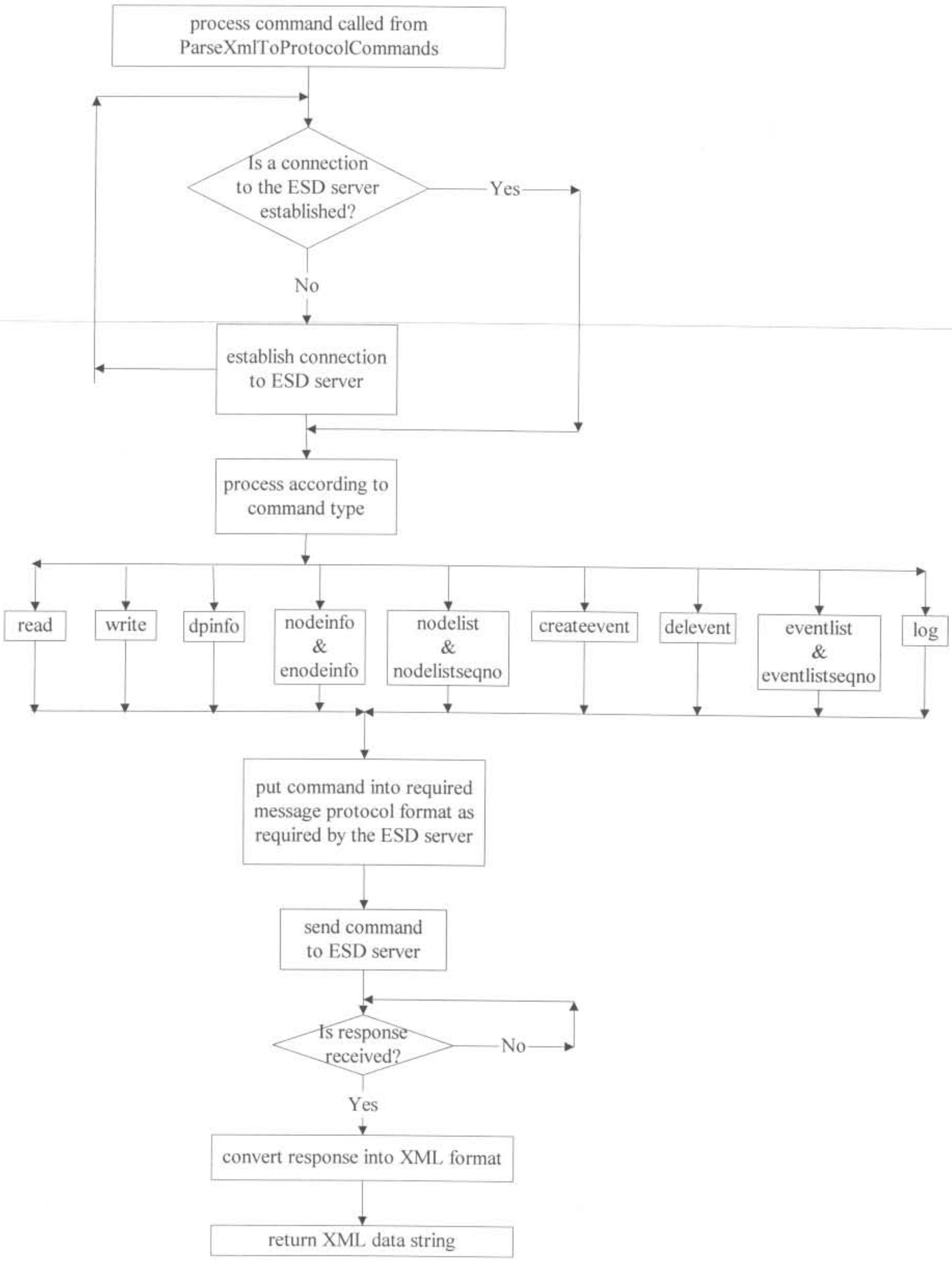

process command called from ParseXmlToProtocolCommands
Is a connection to the ESD server established?
Yes
No
establish connection to ESD server
process according to command type
read
write
dpinfo
nodeinfo & enodeinfo
nodelist & nodelistseqno
createevent
delevent
eventlist & eventlistseqno
log
put command into required message protocol format as required by the ESD server
send command to ESD server
Is response received?
No
Yes
convert response into XML format
return XML data string

Figure 23: Connecting to ESD server and processing commands

Class::ProcessSqlProtocol

This class implements the ProcessProtocolInterface interface. It provides connectivity to the MySQL server application. The ProcessSqlProtocol class uses the LogMessage, the SqlCommander and ProtocolException classes. The class diagram for this class is shown in the figure below.

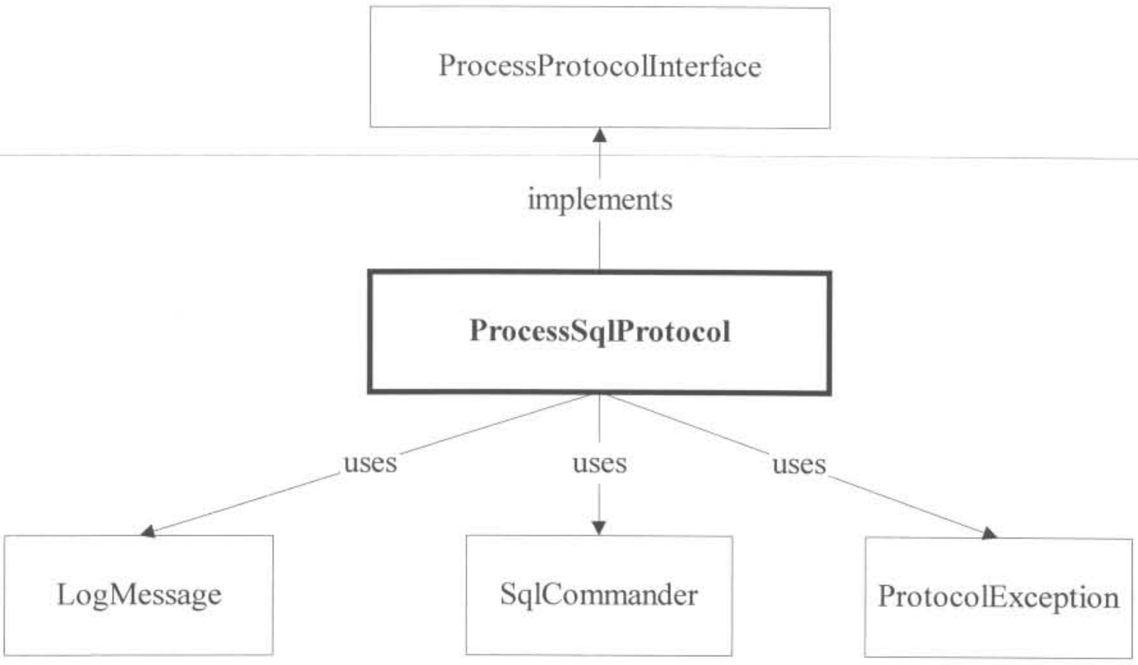


Figure 24: ProcessSqlProtocol class diagram

The main logic flows for this class is shown in the figure below.

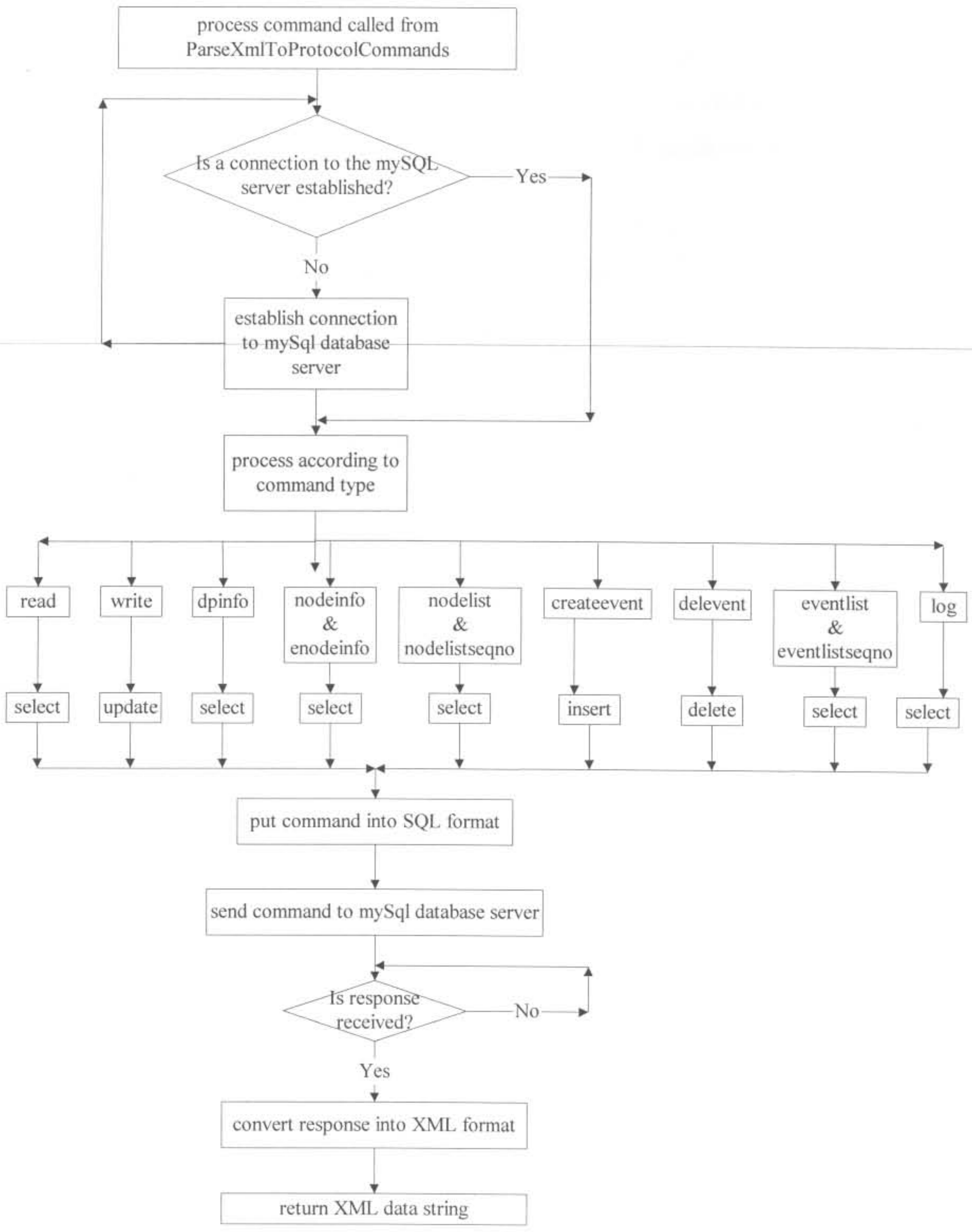

process command called from ParseXmlToProtocolCommands
Is a connection to the mySQL server established?
Yes
No
establish connection to mySql database server
process according to command type
read
write
dpinfo
nodeinfo & enodeinfo
nodelist & nodelistseqno
createevent
delevent
eventlist & eventlistseqno
log
select
update
select
select
select
insert
delete
select
select
put command into SQL format
send command to mySql database server
Is response received?
No
Yes
convert response into XML format
return XML data string

Figure 25: Connecting to MySQL database server and processing commands

Class::ProcessLdapProtocol

This class implements the ProcessProtocolInterface interface. It provides connectivity to the openLDAP server application. The ProcessLdapProtocol class uses the LogMessage, the LdapCommander and ProtocolException classes. The class diagram for this class is shown in the figure below.

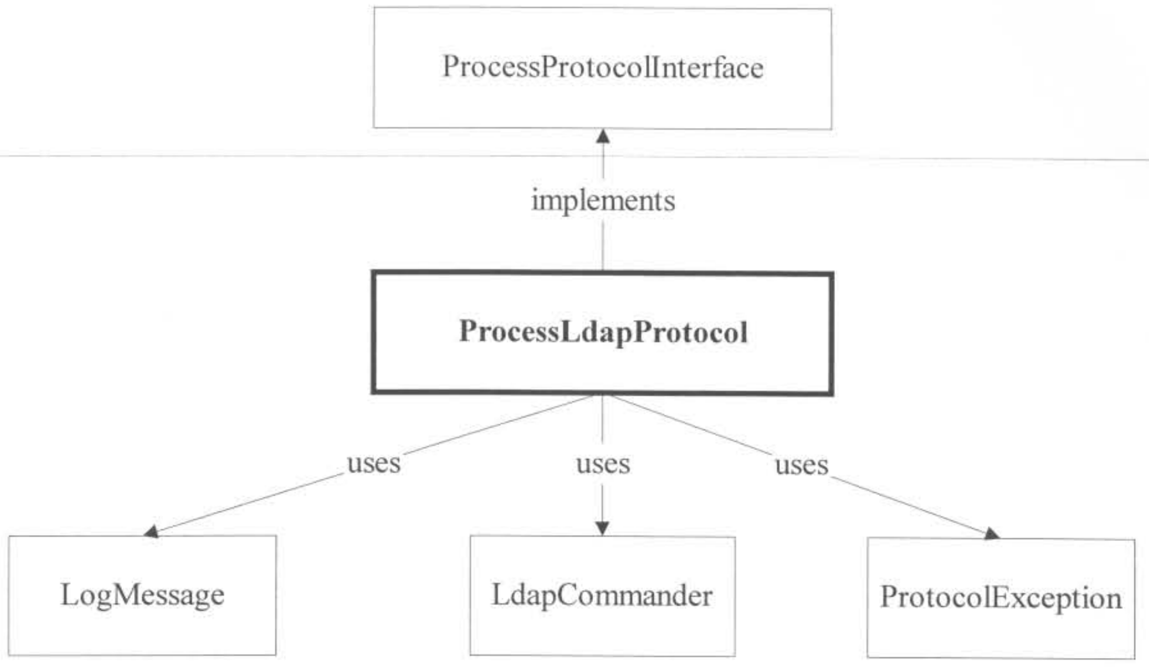


Figure 26: ProcessLdapProtocol class diagram

The main logic flows for this class is shown in the figure below.

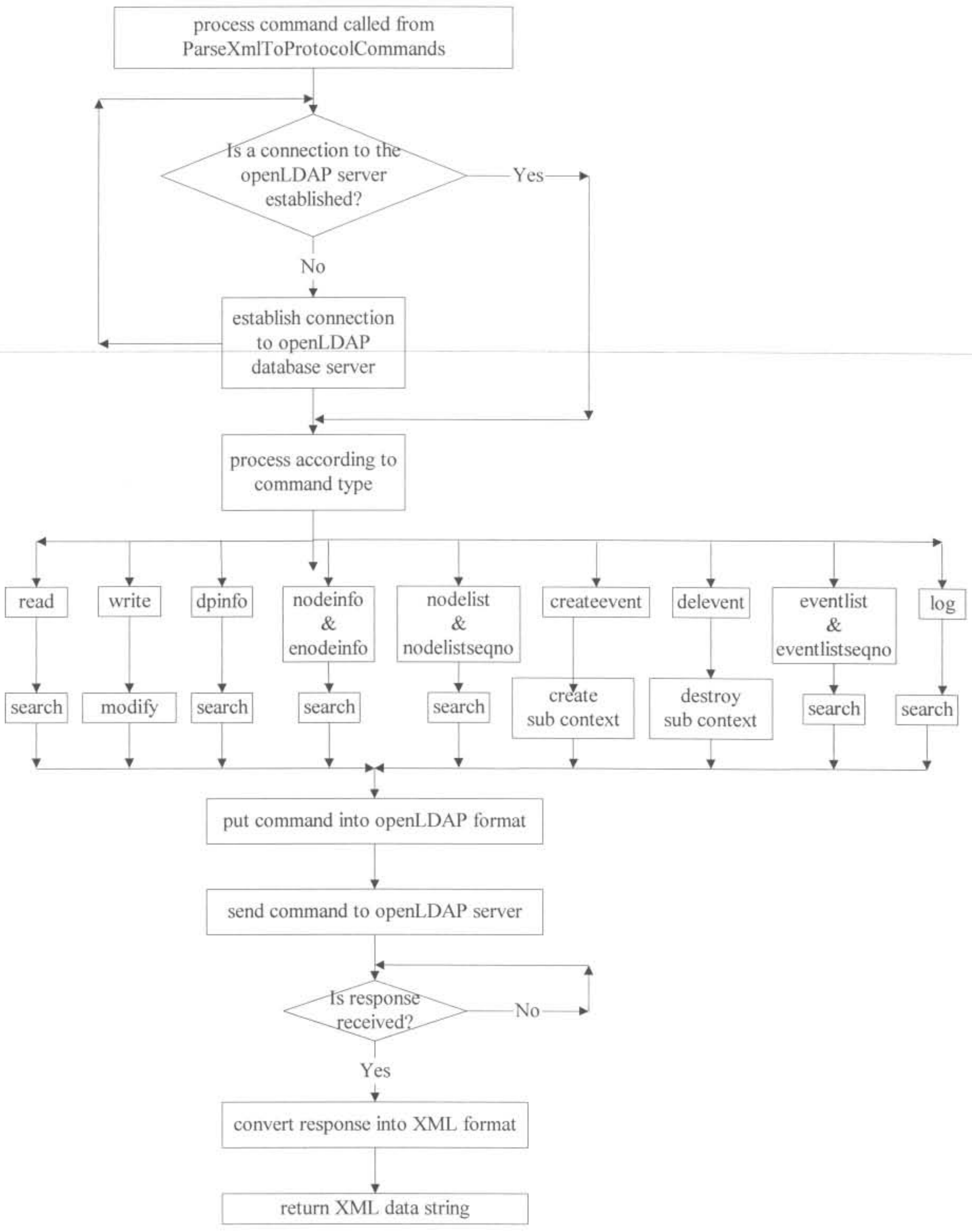

process command called from ParseXmlToProtocolCommands
Is a connection to the openLDAP server established?
Yes
No
establish connection to openLDAP database server
process according to command type
read
write
dpinfo
nodeinfo & enodeinfo
nodelist & nodelistseqno
createevent
delevent
eventlist & eventlistseqno
log
search
modify
search
search
search
create sub context
destroy sub context
search
search
put command into openLDAP format
send command to openLDAP server
Is response received?
No
Yes
convert response into XML format
return XML data string

Figure 27: Connecting to openLDAP server and processing commands

7.2.3 The Data layer

The data layer implements the details of the specific server application functionality. It currently consists of three packages, namely, the esd, sql and ldap package.

7.2.3.1 esd package

The following figure shows the package diagram for esd.

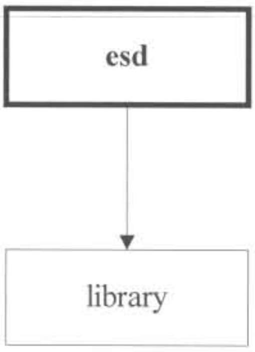


Figure 28: esd package diagram

The esd package contains two classes: EsdCommander and EsdResponse. The EsdResponse class is used by the EsdCommander class.

Class::EsdResponse

The EsdResponse class is used to store the response data returned from the ESD server for the EsdCommander class.

Class::EsdCommander

The EsdCommander class implements the specific functionality to connect to the ESD server, using TCP sockets to send the commands in the required ESD format. The responses are converted into the standard XML format before being returned to the calling function.

The following figure shows the EsdCommander class diagram. The EsdCommander class uses the LogMessage class and the EsdResponse class.

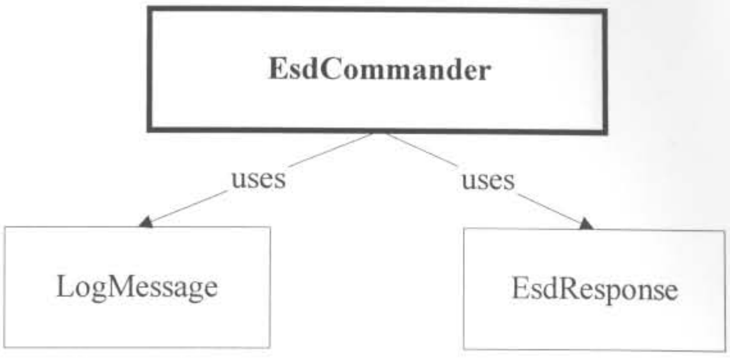


Figure 29: EsdCommander class package

7.2.3.2 sql package

The following figure shows the sql package diagram.

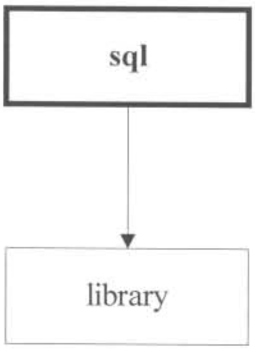


Figure 30: sql package diagram

The sql package contains one class: SqlCommander.

Class::SqlCommander

The SqlCommander class implements the specific functionality to connect to the MySQL database server, using a JDBC driver and to send the commands in the required SQL format. The responses are converted into the standard XML format before being returned to the calling function.

The following figure shows the SqlCommander class diagram. The SqlCommander class uses the LogMessage class.

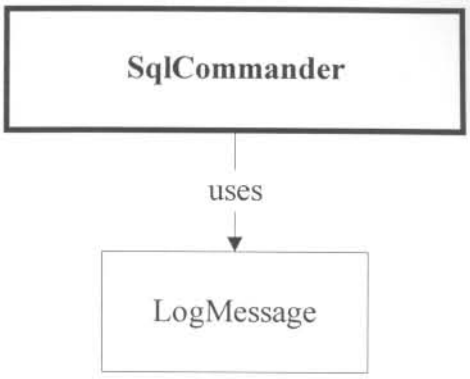


Figure 31: SqlCommander class diagram

7.2.3.3 ldap package

The following figure shows the ldap package diagram.

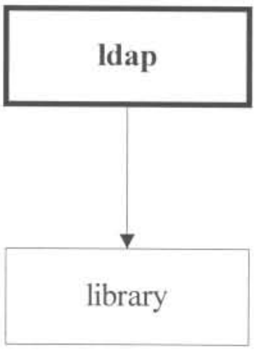


Figure 32: ldap package diagram

The ldap package contains one class: LdapCommander.

Class::LdapCommander

The LdapCommander class implements the specific functionality to connect to the openLDAP server, and to send the commands in the required LDAP format. The responses are converted into the standard XML format before being returned to the calling function.

The following figure shows the LdapCommander class diagram. The LdapCommander class uses the LogMessage class.

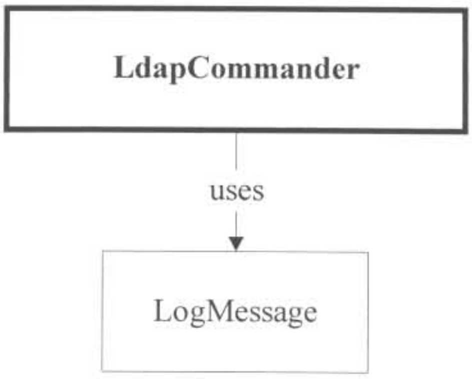


Figure 33: LdapCommander class diagram

The business layer uses the above-mentioned data layer classes, namely, EsdCommander, EsdResponse, SqlCommander and LdapCommander and the logical flow is shown previously with the business layer flowcharts.

7.3 General library classes

The following figure shows the library package diagram. As can be seen from the diagram, the library package has no dependencies on other application packages. It is intended to be used by other packages (refer to previous package diagrams).

7.3.1.1 library package

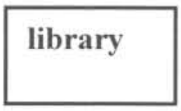


Figure 34: library package diagram

The library package consists of the following classes:

- LogMessage
- CommonDefs
- ClientSocket
- JSPException

LogMessage is used by the other classes, to write messages to the log file in a standard way. It requires the using class to parse it the log file as an input parameter.

The CommonDefs class contains all constant values that are used across the application classes.

The ClientSocket class provides the functionality to open a socket connection.

The JSPException class extends the standard Exception class. It gets thrown if the front-end program detects an error, such as null values parsed as an input parameter.

7.4 Application flowchart

The application uses functions from the various classes described previously. A high level view of the application flowchart is shown below.

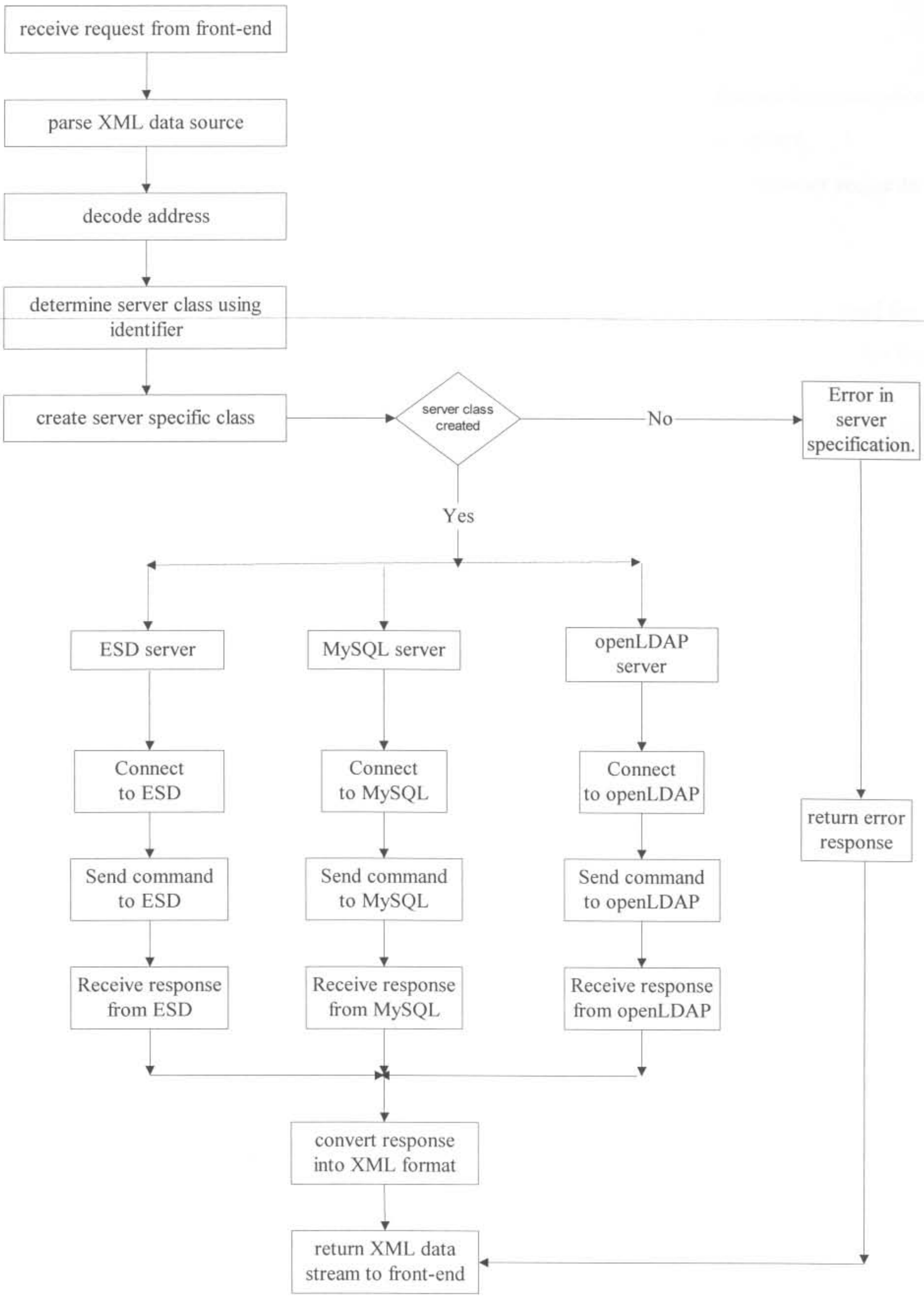

receive request from front-end
parse XML data source
decode address
determine server class using identifier
create server specific class
server class created
No
Error in server specification.
Yes
ESD server
MySQL server
openLDAP server
Connect to ESD
Connect to MySQL
Connect to openLDAP
return error response
Send command to ESD
Send command to MySQL
Send command to openLDAP
Receive response from ESD
Receive response from MySQL
Receive response from openLDAP
convert response into XML format
return XML data stream to front-end

Figure 35: Application flowchart

7.5 Error Handling

There are two classes of exceptions, namely:

- ProtocolException: the server application specific interface class throws this exception, if an error such as being unable to connect to the server application occurs.
- JspException: this exception is thrown if the class that processes the browser requests detects an error, such as null values parsed as an input parameter.

The application also has a log file that logs information to a specified file that can be read for debug purposes. There are different log levels that ensure that the user can set the level for the type of debug messages to be logged to the file. The debug levels are:

Debug Level	Description
DBG_NONE	No debug messages are written to the log file
DBG_EXCEPTION	Exception type debug messages are written to the log file
DBG_ERROR	Class error type messages are written to the log file
DBG_WARNING	Warning type messages are written to the log file
DBG_MESSAGE	A general type of message is written to the log file
DBG_ALL	All types of messages are written to the log file

Table 5: Debug levels

Chapter 8 : RESULTS AND ANALYSIS

This section shows the measurements of the time taken to send and receive non-XML and XML type messages to the different application servers. All measurements are in milliseconds. The results also show the differences between the different server applications in processing request-response type messages.

8.1 Results

It should be noted that these time values are not optimal as there are a number of Input/Output operations (such as writing messages to a log file and/or standard output) that occur during the processing of each command. Because these operations are similar across application implementations, no significant time benefit is accrued to any application server implementation. However, the results are not meant to reflect the optimum performance levels of each application server.

8.1.1 ESD Results

The following table shows the result of time taken (in milliseconds) to send a command to an ESD server if:

1. The command is not enclosed in XML
2. The command is enclosed in XML and needs to be parsed.

Command	ESD No XML (ms)	ESD with XML (ms)
READ	12	522
WRITE	5	172
DPINFO	3	237
NODEINFO	4	270
ENODEINFO (47 data points)	97	8590
NODELIST	8	143
NODELISTSEQNO	39	55
REFRESHNODELIST	3	50
UPDATENODELIST	32	40

CREATEEVENT	108	193
DELETEEVENT	117	638
EVENTLIST	43	125
EVENTLISTSEQNO	5	45
LOG	36	59
VERSION	9	62

Table 6: No XML vs. XML for ESD

8.1.2 SQL Results

The following table shows the result of time taken (in milliseconds) to send a command to a SQL server (i.e. MySQL database server) if:

1. The command is not enclosed in XML
2. The command is enclosed in XML and needs to be parsed.

Command	MySQL No XML (ms)	MySQL with XML (ms)
READ	19	406
WRITE	19	228
DPINFO	17	285
NODEINFO	10	210
ENODEINFO (4 data points)	88	368
NODELIST	4	419
NODELISTSEQNO	7	164
CREATEEVENT	4	223
DELETEEVENT	1	153
EVENTLIST	7	126
EVENTLISTSEQNO	2	207
LOG	20	102

Table 7: No XML vs. XML for MySQL

As mentioned in section 4.5 not all commands are implemented in the MySQL and openLDAP implementations.

Note, the fact that the enodeinfo command with XML is significantly larger in ESD compared to SQL is because there are 47 data points in the ESD implementation, as compared to the MySQL implementation, which only has 4 data points for the particular node.

8.1.3 LDAP Results

The following table shows the result of time taken (in milliseconds) to send a command to the openLDAP server if:

1. The command is not enclosed in XML
2. The command is enclosed in XML and needs to be parsed.

Command	**openLDAP No XML (ms)**	**openLDAP with XML (ms)**
READ	141	495
WRITE	138	430
DPINFO	125	377
NODEINFO	186	442
CREATEEVENT	264	530
DELETEEVENT	136	485
LOG	144	427

Table 8: No XML vs. XML for openLDAP

8.1.4 Results of different application servers

It is interesting to compare the times for the different protocols without XML and with XML

The following table shows the result of time taken (in milliseconds) to send and receive a non-XML message to each of the different application servers.

Command	ESD	MySQL	OpenLDAP
READ	12	19	141
WRITE	5	19	138
DPINFO	3	17	125
NODEINFO	4	10	186
CREATEEVENT	108	4	264
DELETEEVENT	117	1	136
LOG	36	20	144

Table 9: Time difference for application servers without XML

The following table shows the result of time taken (in milliseconds) to send and receive an XML message to each of the different application servers.

Command	ESD	MySQL	openLDAP
READ	522	406	495
WRITE	172	228	430
DPINFO	237	285	377
NODEINFO	270	210	442
CREATEEVENT	193	223	530
DELETEEVENT	638	153	485
LOG	59	102	427

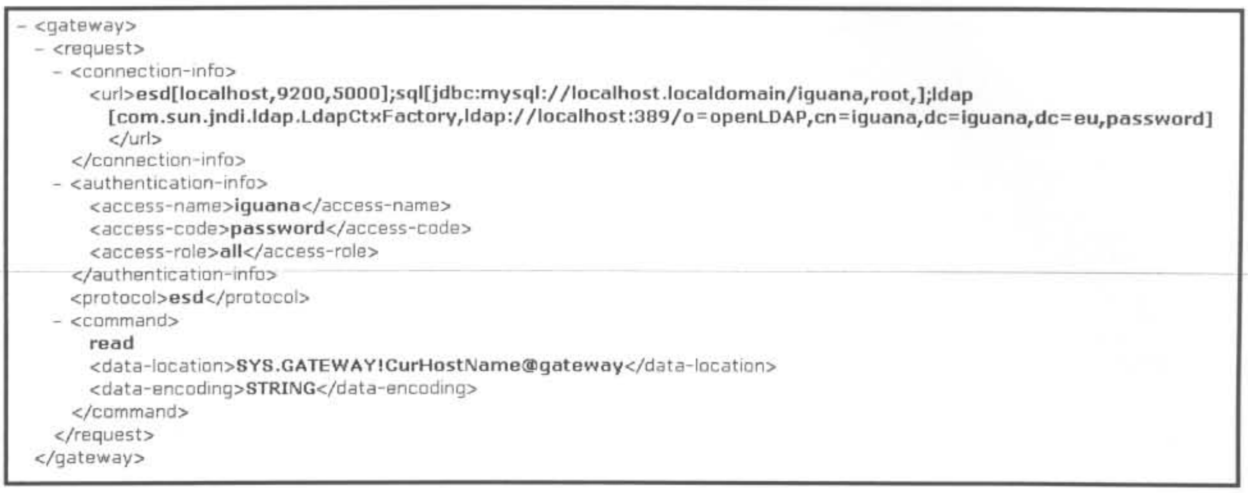
Table 10: Time difference for application servers with XML

8.1.5 Measuring the relationship between number of XML messages and time.

The time measurement per XML request and per XML command was taken to determine if the relationship between the number of XML messages and the time taken to process them is linearly proportional.

The command used to perform the measurements was the "READ" command sent to the ESD server. The messages sent were of the following types:

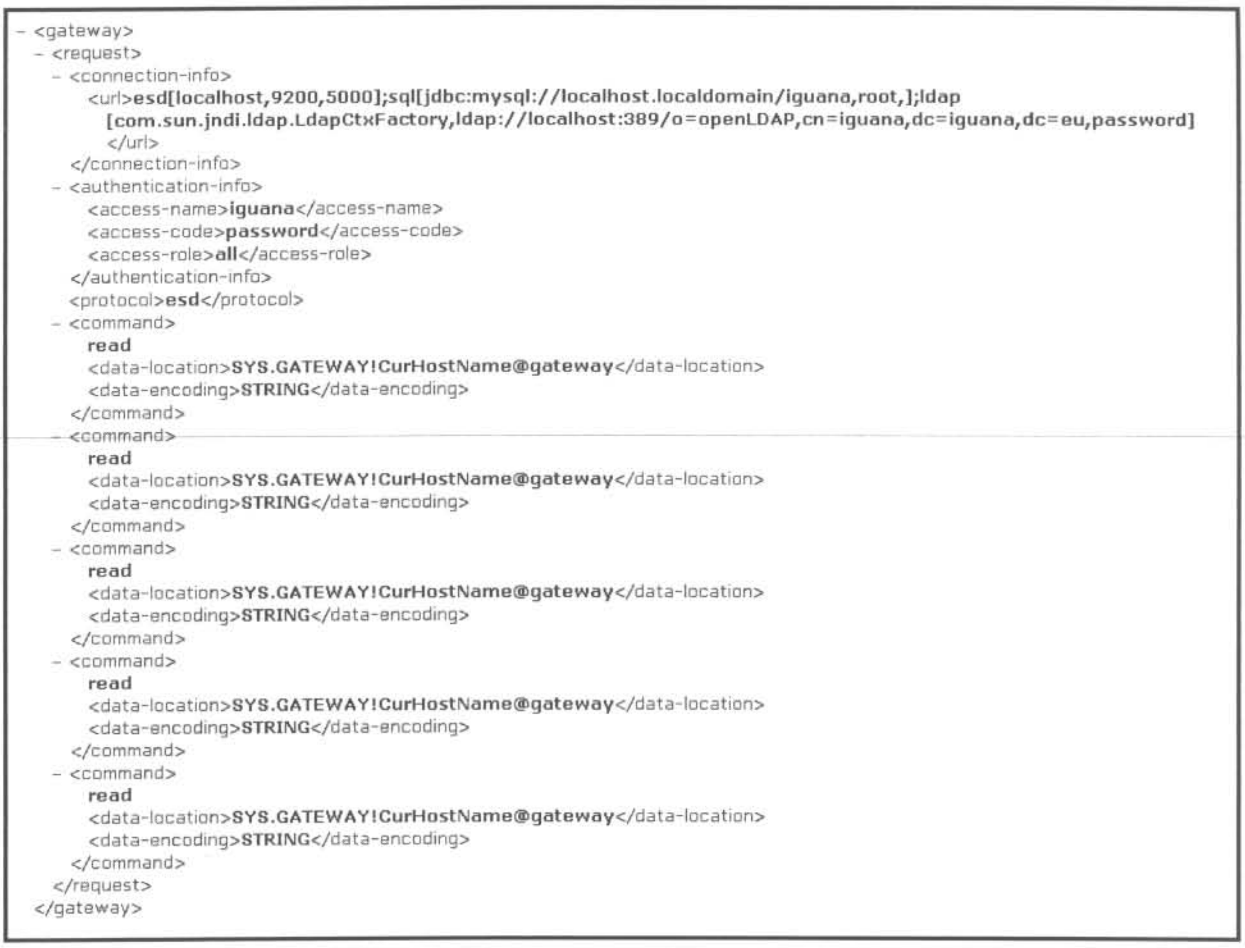
1. Multiple individual request commands as shown in Figure 35, or
2. A single request containing multiple read commands as shown in Figure36.



```
- <gateway>
  - <request>
    - <connection-info>
        <url>esd[localhost,9200,5000];sql[jdbc:mysql://localhost.localdomain/iguana,root,];ldap
          [com.sun.jndi.ldap.LdapCtxFactory,ldap://localhost:389/o=openLDAP,cn=iguana,dc=iguana,dc=eu,password]
          </url>
      </connection-info>
    - <authentication-info>
        <access-name>iguana</access-name>
        <access-code>password</access-code>
        <access-role>all</access-role>
      </authentication-info>
      <protocol>esd</protocol>
    - <command>
        read
        <data-location>SYS.GATEWAY!CurHostName@gateway</data-location>
        <data-encoding>STRING</data-encoding>
      </command>
    </request>
  </gateway>
```

Figure 36: Request and single command sent multiple times

The XML message shown in Figure 35 was sent multiple times to the ESD server and the time taken to process the XML messages was measured.



```
- <gateway>
  - <request>
    - <connection-info>
        <url>esd[localhost,9200,5000];sql[jdbc:mysql://localhost.localdomain/iguana,root,];ldap
          [com.sun.jndi.ldap.LdapCtxFactory,ldap://localhost:389/o=openLDAP,cn=iguana,dc=iguana,dc=eu,password]
          </url>
      </connection-info>
    - <authentication-info>
        <access-name>iguana</access-name>
        <access-code>password</access-code>
        <access-role>all</access-role>
      </authentication-info>
      <protocol>esd</protocol>
    - <command>
        read
        <data-location>SYS.GATEWAY!CurHostName@gateway</data-location>
        <data-encoding>STRING</data-encoding>
      </command>
    - <command>
        read
        <data-location>SYS.GATEWAY!CurHostName@gateway</data-location>
        <data-encoding>STRING</data-encoding>
      </command>
    - <command>
        read
        <data-location>SYS.GATEWAY!CurHostName@gateway</data-location>
        <data-encoding>STRING</data-encoding>
      </command>
    - <command>
        read
        <data-location>SYS.GATEWAY!CurHostName@gateway</data-location>
        <data-encoding>STRING</data-encoding>
      </command>
    - <command>
        read
        <data-location>SYS.GATEWAY!CurHostName@gateway</data-location>
        <data-encoding>STRING</data-encoding>
      </command>
    </request>
  </gateway>
```

Figure 37: Request with multiple commands sent once off

The single XML message with multiple read commands shown in Figure 36 was sent to the ESD server and the time taken to process the message was measured.

The measured time taken to process the XML message(s) are shown in Table 11.

Number of messages	Request with single command (msec)	Time per XML message for multiple request-command pairs (msec)	Request with multiple commands (msec)	Time per XML message for single request-multiple commands (msec)
1	211	211	217	217
2	270	135	257	129
3	325	108	276	92
4	399	100	307	77
5	480	96	333	67
6	618	103	479	80
7	715	102	555	79
8	691	86	649	81

Number of messages	Request with single command (msec)	Time per XML message for multiple request-command pairs (msec)	Request with multiple commands (msec)	Time per XML message for single request-multiple commands (msec)
9	860	96	807	90
10	929	93	777	78
15	1432	95	1287	86
20	1777	89	1981	99
25	2488	100	2340	94
30	3048	102	2939	98
35	3650	104	3455	99
40	4106	103	3940	99
50	4854	97	5012	100
60	6895	115	6080	101
70	7045	101	7200	103
80	8455	106	8071	101
90	9486	105	9056	101
100	10349	103	10314	103

Table 11: Time taken to process Single and Multiple command XML messages

By plotting the above values on a line graph the following results were obtained.

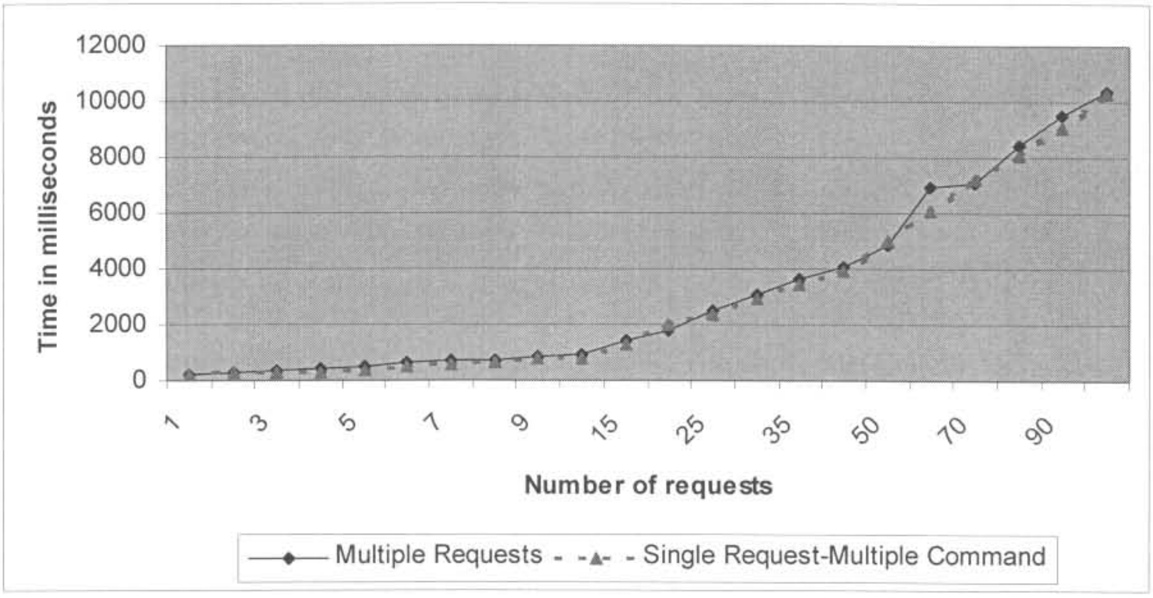


Figure 38: Graphical representation of number of XML messages and time

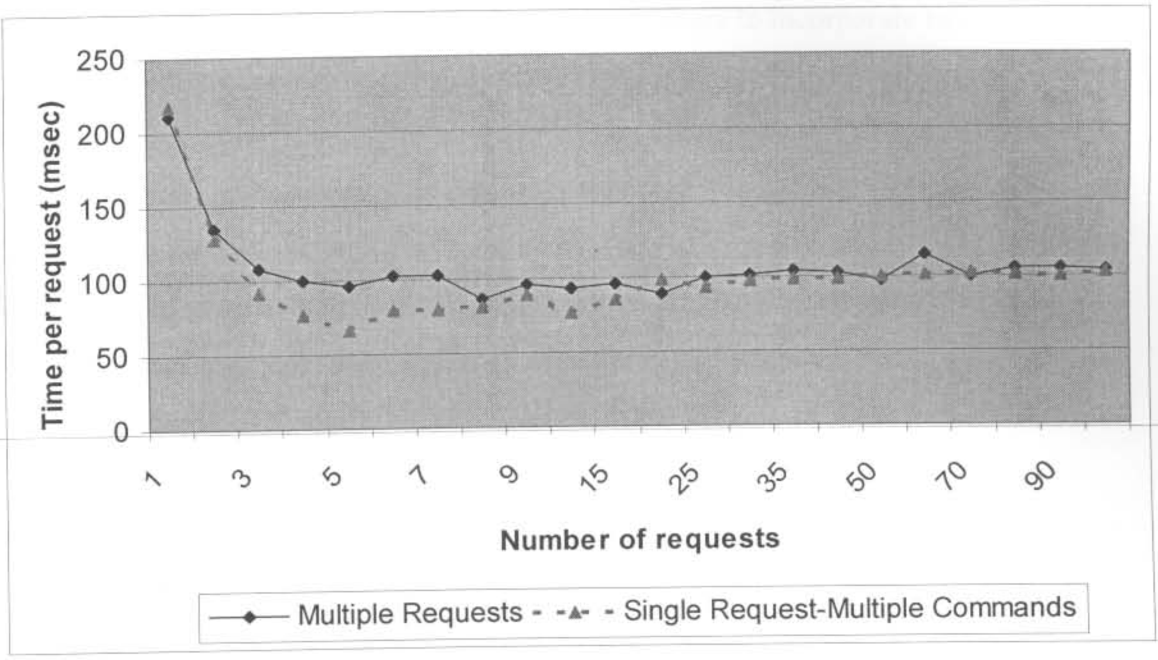


Figure 39: Graphical representation of time taken to process an XML command as the number of commands increase

8.2 Analysis of Results

8.2.1 Comparison of XML vs. non-XML type messages

From the above results (Table 6, Table 7, and Table 8), it is clear that non-XML type messages are faster and provide better performance results than XML type messages. It was known in advance that transforming messages into XML format would increase the processing overhead because of the inherent verbosity of XML. The additional time is due to the construction of messages into XML format and the parsing of the XML messages to determine the type of application server.

There is clearly a trade-off between performance (i.e. the amount of time it takes to send a request and receive a response) and the flexibility of providing a common standard message interface to multiple applications that comes with using XML.

The following advantages of using XML in the implementation were identified:

- Standard message format for multiple applications, i.e. XML is application independent.
- Common gateway handles requests to multiple application servers.

- Flexibility in extending the XML document structure to incorporate new application servers with minimum additional changes to existing coding infrastructure.
- Additional security benefit of having only one access point (HTTP port) to multiple applications made available to external networks.

The following disadvantages of using XML in the implementation were identified:

- Slower response times, leading to decreased performance
- Increased CPU usage
- Increased memory resource usage

8.2.2 Comparison of application server performance with respect to XML vs. non-XML type messages.

It is interesting to look at the difference in performance levels across the different application servers even though it is not a requirement of this research topic. It would appear that ESD has been optimised for "read" type of operations, as shown by the better performance in reading data point and node information.

The database implementation has better performance when it comes to inserting new data and deleting data. The directory server implementation (i.e. openLDAP) is the worst performer in terms of response times, although LDAP implementations are assumed to be better optimised for read type operations compared to databases (Section 2.1.3).

Note, the changes in performance between ESD and MySQL when XML formatting is imposed on the messages. The most probable explanation for this is that the ESD response for non-XML messages is a string that is sent back to the test program without additional formatting, whereas the SQL implementation has to read the result set and put the response into a string format before returning the response to the test program.

For XML type messages, the ESD response string has to be processed according to the predefined ESD protocol (as described in [3]) for each command response and put into the XML format.

Therefore it would appear that if any interpretation of the ESD response according to the protocol specification were required then its response times might be similar to the SQL implementation. Again, the LDAP implementation is the worst performing "protocol".

8.2.3 Analysis of time taken to process multiple XML messages

As expected and as shown in Figure 38, the time taken to process the messages increases as the number of messages increase. However the time taken to process a message does not increase in a directly proportional manner, i.e. the time taken for two messages is not double the time taken for one message etc. as shown in Figure 39. The time taken, as the number of messages increases appears to reach a constant plateau where the average time taken per request/command tends to a constant level of around +-100 milliseconds per request/command.

The time taken to process a single request with multiple commands within the request element is slightly smaller then the time taken to process multiple request-command messages. This is because connection-info, request and other higher order elements do not have to be parsed for each command. Because the number of elements preceding a command element is small, the time taken before reaching the command element is relatively small. This indicates that the XML parser is efficient and may be able to handle larger XML documents with minimal additional performance cost.

From the previous results, we can conclude that as the number of commands increases (i.e. exceeds three requests/commands), the time taken to process each command increases in a linear relationship that approximates the function $y = 100 x$ (for single request-single commands type messages) and $y = 91 x$ (for single request-multiple command type messages).

It should be noted that these measurements were done on a single computer, i.e. no network transmission overheads affect the results. The choice of the appropriate XML message format must take into account the size of the message and the transmission delays of sending data across a network.

Chapter 9 : CONCLUSION

An XML document model for sending and receiving messages between heterogeneous server applications was designed and implemented. The modelling of the XML document required evaluating client-server applications and determining an optimum method of communicating with each application using a standard message format.

The XML model achieves this by using the location of the required data to determine which application server the message is intended for. The upside of this is that a single user interface can be used to send and receive messages to multiple application servers. The downside is that the user has to enter in a unique application identifier when entering in the location of the data.

An alternative method is to include an additional element in the XML request document that describes the application server the message is intended for. The upside of this solution is that the user is not required to specify a unique identifier per application. However, the downside is that there has to be a separate user interface per application server such that when it formats a message request, the type of application server the message is intended for is added as the appropriate protocol element value.

The possibility of allowing the gateway to identify the application server was also developed but this solution proved inadequate if more than one application server was active at the time because the message would be sent to the first active server, even though it was not intended for that server. The only time this solution would be appropriate is when data from multiple application servers needs to be collected and collated in a single response. Then the message can be sent to all active servers and combined into a single response message.

The application was designed to be easily expandable. This is achieved by making use of the properties functionality provided by Java that enables a program to read a file that stores data as key-value pairs. Therefore each application server is assigned a unique identifier (such as ESD) and the identifier is related to the class that processes messages intended for that server.

Therefore, new application servers can be easily added to the implementation by providing a new class that provides access to the new server application and updating the properties file to include a key-value pair that associates the unique identifier of the application server with the class that will process its specific commands.

The use of XML as the data description language and integration mechanism, where the XML parser is used as the "translator" between the description of the data and different systems validates the conclusions drawn by Blattner et al. [18,19] that a "parser-generator" would be most suitable for generic message translation.

There appears to be no similar XML model for client-server message interaction developed previously. Bi et al. [15] developed an XML model for interaction with legacy applications but did not focus on the development of a common message interface that could be used across multiple client-server applications. The developed XML model can therefore be considered to be a novel solution.

After the design was implemented and tested, the performance of XML and non-XML messages were evaluated. As expected the increased verbosity of XML results in a larger footprint that requires more processing time and resources. This means that any implementation using XML has to carefully weigh the benefits of flexibility, extensibility and standard message formats against reduced performance. This conforms to the conclusions of the study conducted by Boedjang et al. [21] that the performance measurements of applications that run application-specific code are faster than those that use generic message passing software.

XML is currently not suitable for applications that require high-speed real time responses. Client applications that use the Internet to obtain server information from multiple applications will benefit from reduced client side complexity. Server applications that serve large client bases and therefore require smaller resource allocation per request may not be scalable because of the integration with XML. The reduced performance levels from using XML means it does not scale to handle large numbers of concurrent client requests.

However, if the applications are run in batch mode where the need for fast (i.e. micro and millisecond) responses are not important, then the solution is useful. As long as the time delay is not too long in human (user) terms (and in this implementation it is not noticeable as responses are less than or within seconds) then the additional response times caused by the XML footprint is negligible. Therefore, it can be concluded that when used to encode messages in a standard format for use in client-server type environments requiring human-computer interaction or batch processing, that XML can provide significant advantages.

Chapter 10 : REFERENCES

1. H. Johner, L. Brown, F. Hinner, W. Reis, J. Westman, Understanding LDAP, http://www.redbooks.ibm.com.
2. M. Gertz, Oracle/SQL Tutorial 1, http://www.db.cs.ucdavis.edu.
3. M. Lobachov, Communication to the Extended Service Daemon (esd), Iguana project documentation. (ESD daemon proto.txt).
4. M. Lobachov, Event Language, Iguana project documentation. (event language.txt).
5. H. Kleiner, O. Triebl, Implementing LDAP connectivity for the Iguana Project.
6. A. Nahimovsky; T. Myers, XML Programming, Apress, 2002.
7. E. Armstrong, S. Bodoff, D. Carson, M. Fisher, D. Green, K. Haase, The Java™ Web Services Tutorial, August 2002.
8. B. Marchal, XML by Example, Que, 2000.
9. Directory Services Markup Language v2.0, http://www.oasis-open.org/committess/dsml/docs.
10. "SQL-XML Group Picks INCITS to Develop Standards.htm", www.incits.org.
11. A. Eisenberg and J. Melton, SQL/XML and the SQLX Informal Group of Companies, http://www.acm.org/sigmod/record/issues/0109/standards.pdf.
12. An XML vocabulary for CIM Management Information, http://www.dnmtf.org/standards/xmlw.php.
13. T. Goddard, Towards XML Based Management and Configuration, Internet-Draft.
14. J. Jaworski, Java2 Certification Training Guide, 1999.
15. Y. Bi, M.E.C. Hull, P.N. Nicholl, An XML approach for legacy code reuse, The Journal of Systems and Software, 2002, Pages: 77 - 89.
16. K.L.E. Law XML on LDAP Network Database, IEEE Canadian Conference on Electrical and Computer Engineering, 2000.
17. D. Lewis and J.D. Mouritzsen, The role of XML in TMN evolution, IEEE International Symposium of Integrated Network Management Proceedings, 2001.
18. M. Blattner, L. Kou, J. Carlson, D. Daniel, A Visual Interface for Generic Message Translation, IEEE Workshop on Visual Languages, 1988, Page(s): 121 –126.

19. M. Blattner and L. Kou, A User Interface for Computer-Based Message Translation, IEEE Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences, 1989. Vol.IV: Emerging Technologies and Applications Track.
20. P. Peinl and B. Mitschang, Towards an integrated Systems Approach for Mobile Traveller Applications, IEEE First International Conference on Web Information Systems Engineering (WISE'00)-Volume 1, June 19 - 20, 2000.
21. R. Bhoedjang, J. Romein, H. Bal, Optimizing Distributed Data Structures Using Application-Specific Network Interface Software, IEEE Proceedings. 1998 International Conference on Parallel Processing, 1998, Page(s): 485 –492.
22. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler (Editors), Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation, October 2000, http://www.w3.org/TR/REC-xml.
23. D. C. Fallside (Editor), XML Schema Part 0: Primer, W3C Recommendation, May 2001, http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/.
24. The OpenLDAP Project http://www.openldap.org.
25. The OpenLDAP Administrator's Guide, http://www.openldap.org/doc/admin.
26. M. Wahl, T. Howes, S. Kille, Lightweight Directory Access Protocol (version 3), 1997, RFC 2251, www.ietf.org.
27. V. Ryan, S. Seligman, R. Lee, Schema for Representing Java(tm) Objects in an LDAP Directory, October 1999, RFC 2713, www.ietf.org.
28. R. L. Costello (coordinator), XML Schemas: Best Practices, 2003, http://www.xfront.com/BestPracticesHomepage.html.

Addendum A: IGUANA Structured Query Language Daemon (ISQLD)

The database schema used is similar to the IGUANA Structured Query Language Daemon (ISQLD). The database tables are:

1. NODES
2. DATAPOINTS
3. LOGS
4. EVENTS

1. NODE TABLE

Field	Description
NodeAddress	FANTYPE.FANID!FanNodeAddress
FANTYPE	control network type
FANID	unique symbolic name of a FAN daemon
FanNodeAddresss	symbolic name of the node
dp_num	number of data points on this node
nsid_str	self identification string
opt1_str	optional parameters for LonWorks (node location)
opt2_str	optional parameters for LonWorks (program ID)
Error	Error code for this node, received from ESD or, 0 on success

2. DATAPOINTS TABLE

Field	Description
NodeAddress	FANTYPE.FANID!FanNodeAddress
DpAddress	data point address
DataType	FAN-specific data type of the data point
Encoding	encoding of the data
Access	accessibility of this data point
dp_name	data point name
sid_string	self identification string
Data	data value
Error	Error code for this datapoint received from ESD, or 0 on success

3. LOGS TABLE

Field	Description
SQLEventID	unique ID
LogLineNum	line number of this log entry
Timestamp	timestamp
LogData	the log entry data
Error	Error code for this datapoint received from ESD, or 0 on success

4. EVENTS TABLE

Field	Description
SQLEventID	unique ID generated by the client
EventCriteria	event criteria
EventAction	event action
EventDescription	event description

Addendum B: IGUANA LDAP Schema

The IGUANA LDAP schema describes the objectclass and the attributes of that objectclass. The following tree structure describes the schema, starting with the object class and its attributes.

- ObjectClass = iguanaFAN
  - Attributes
    - IguanaFanID
    - IguanaFanDaemonID
    - IguanaFanType
    - iguanaDescription

- ObjectClass = iguanaNode
  - Attributes
    - iguanaNodeAddress
    - iguanaFanNodeAddress
    - iguanaFanDpNum
    - iguanaLocationString
    - iguanaProgramID

- ObjectClass = iguanaDp
  - Attributes
    - iguanaDpAddress
    - iguanaDpId
    - iguanaEncoding
    - iguanaSupportedEncodings
    - iguanaDataType
    - iguanaAccess
    - iguanaName
    - iguanaValue

- ObjectClass = iguanaEvent
  - Attributes
    - iguanaEventID
    - iguanaEventCriteria
    - iguanaAction
    - iguanaICCPrivate

- ObjectClass = iguanaLog
  - Attributes
    - iguanaLineNum
    - iguanaTimeStamp
    - iguanaLogData

Contact Information

Postal Address	P.O. Box 1337, Roosevelt Park, 2129
E-mail	schinnappen@postino.up.ac.za
Tel number	+27-12-420-4335
Cell number	+27-84-580-3226