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].

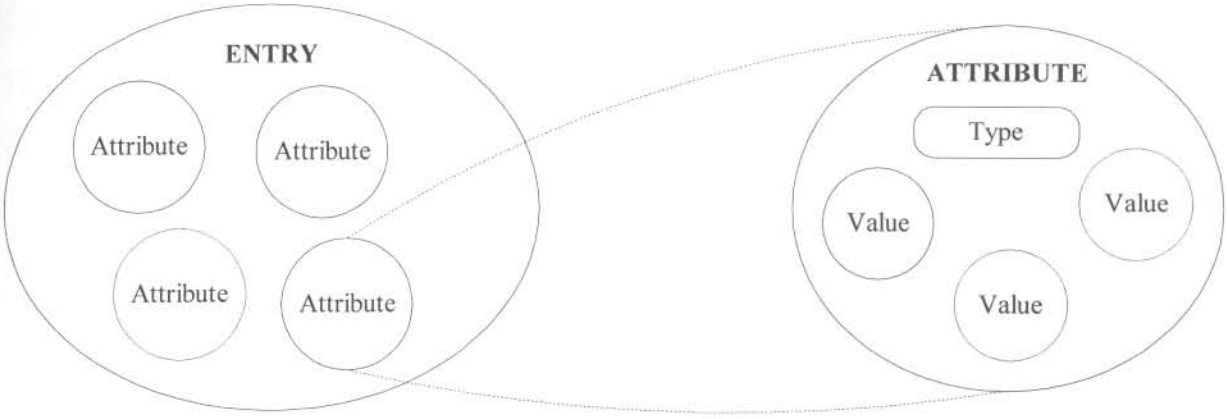

Figure 3: Entries, Attributes and Values [1]

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:

Column 1	Column 2	…	Column n
			← Record (or Tuple)
…	…	…	…

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.

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.

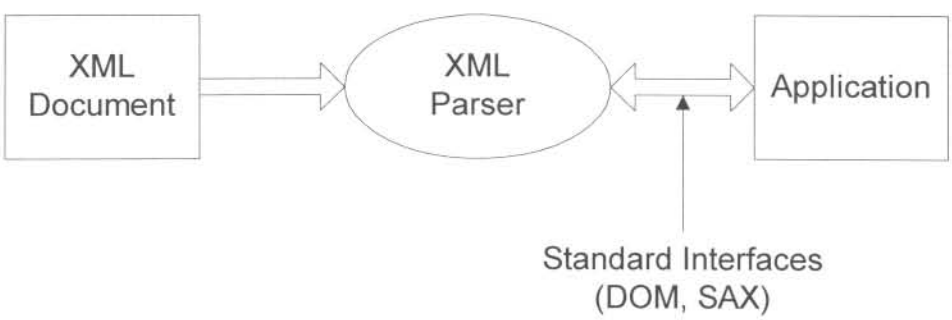

Figure 5 : XML data, the parser, and the application [6]

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.

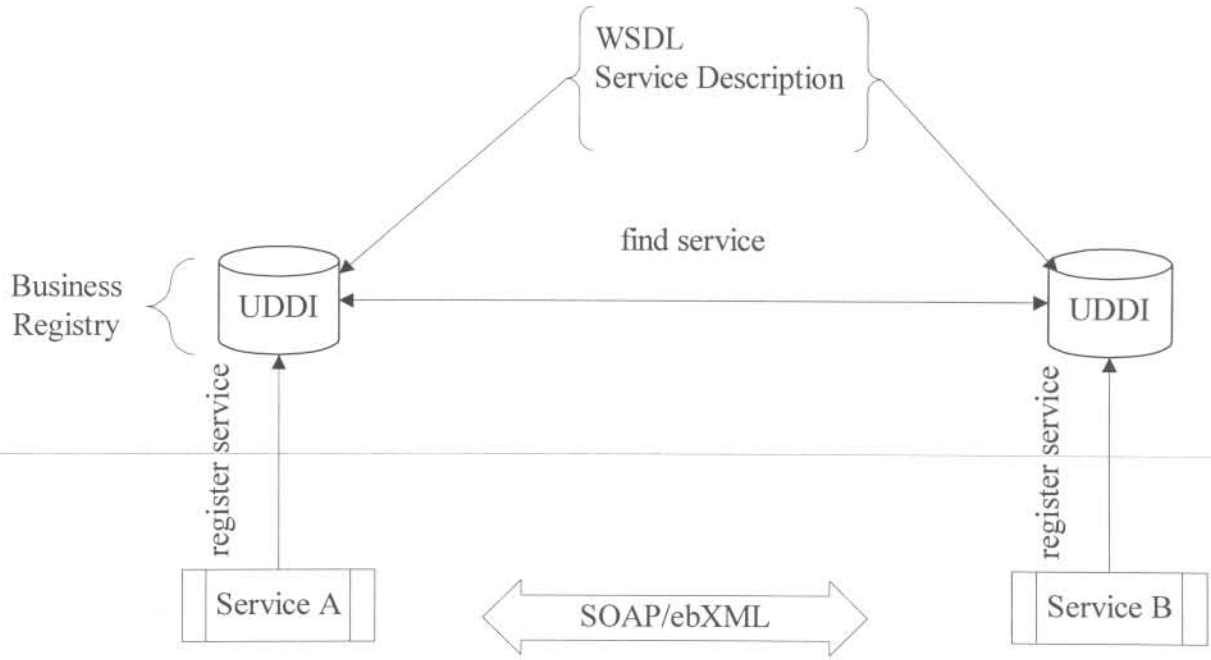

Figure 6: Overview of Web Services

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.