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:

University of Pretoria etd – Rimer, S C (2006)

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).

University of Pretoria etd – Rimer, S C (2006)

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

University of Pretoria etd – Rimer, S C (2006)

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.

University of Pretoria etd – Rimer, S C (2006)

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-locatio
        <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.

University of Pretoria etd – Rimer, S C (2006)

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.

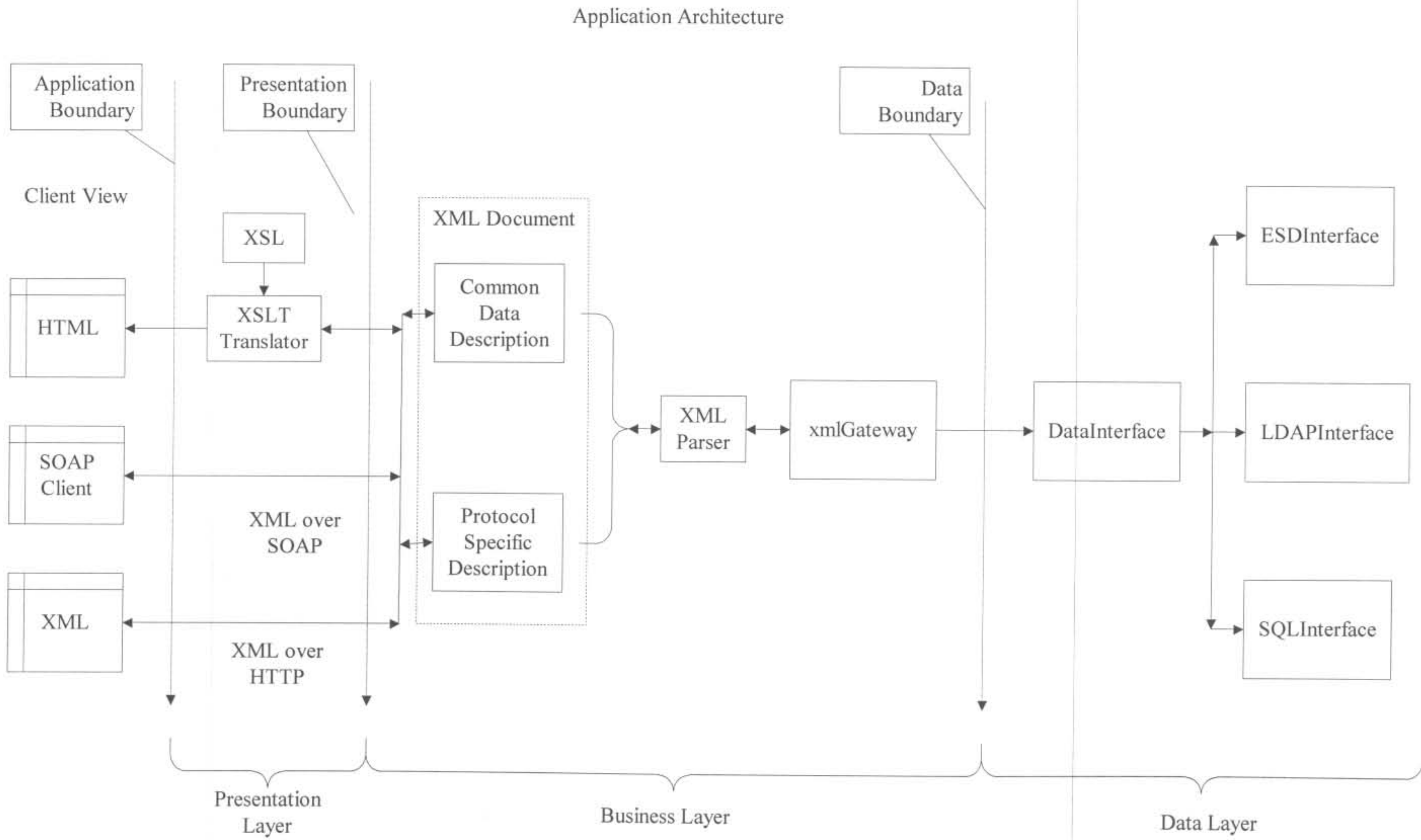

Figure 10: Application Architecture

University of Pretoria etd – Rimer, S C (2006)

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.

University of Pretoria etd – Rimer, S C (2006)

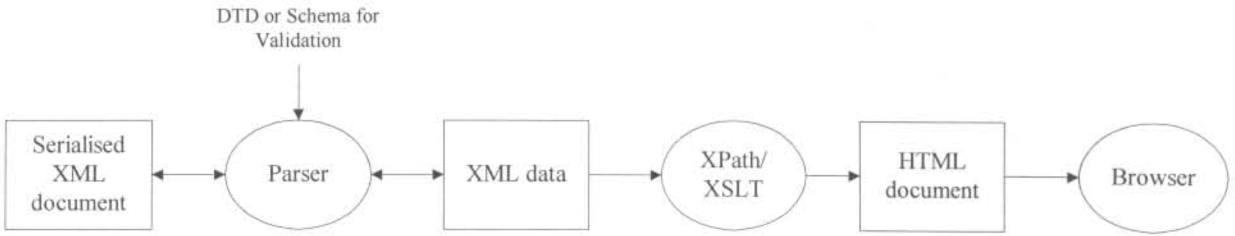

Figure 11: XML processing for browser

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.

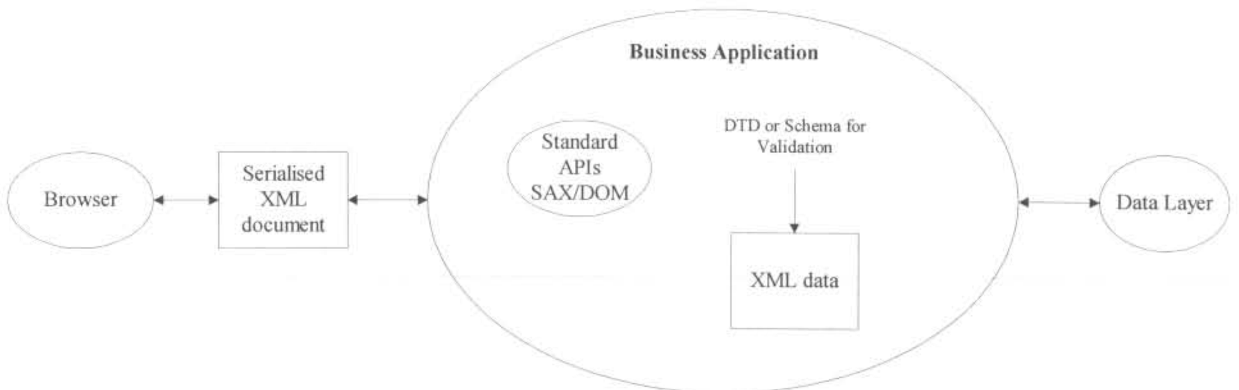

Figure 12: Representation of the business layer

University of Pretoria etd – Rimer, S C (2006)

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.

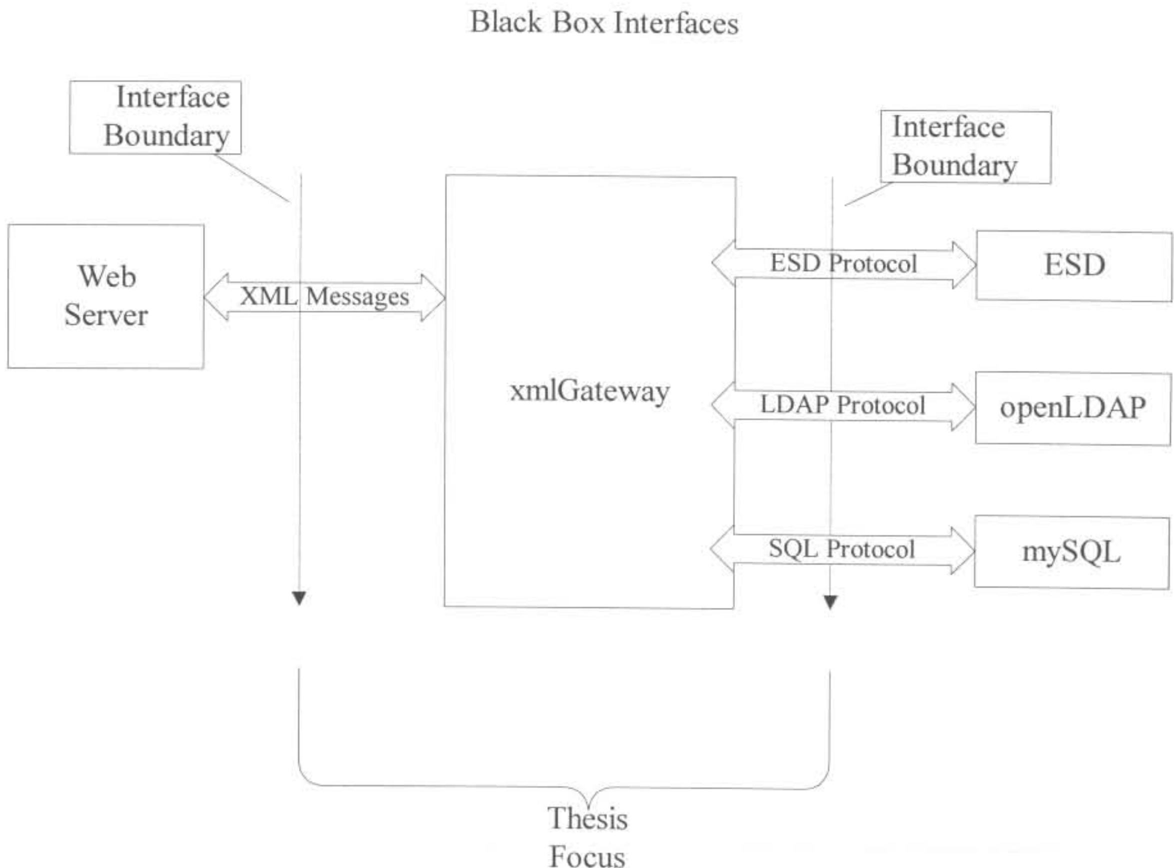

Figure 13: External Interfaces

University of Pretoria etd – Rimer, S C (2006)

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.