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.

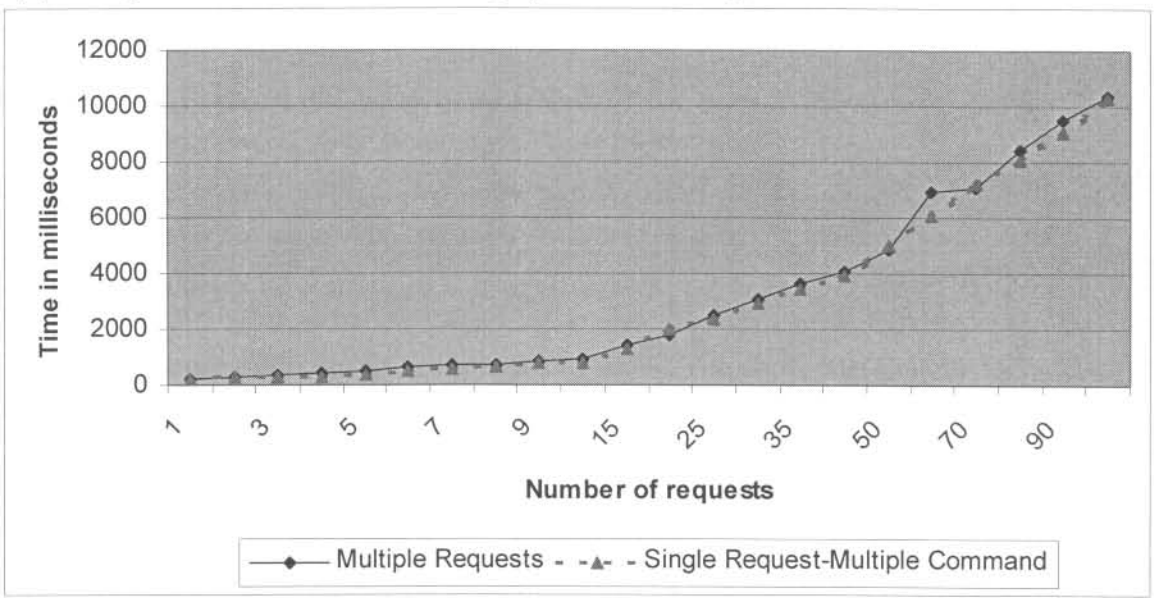

Figure 38: Graphical representation of number of XML messages and time

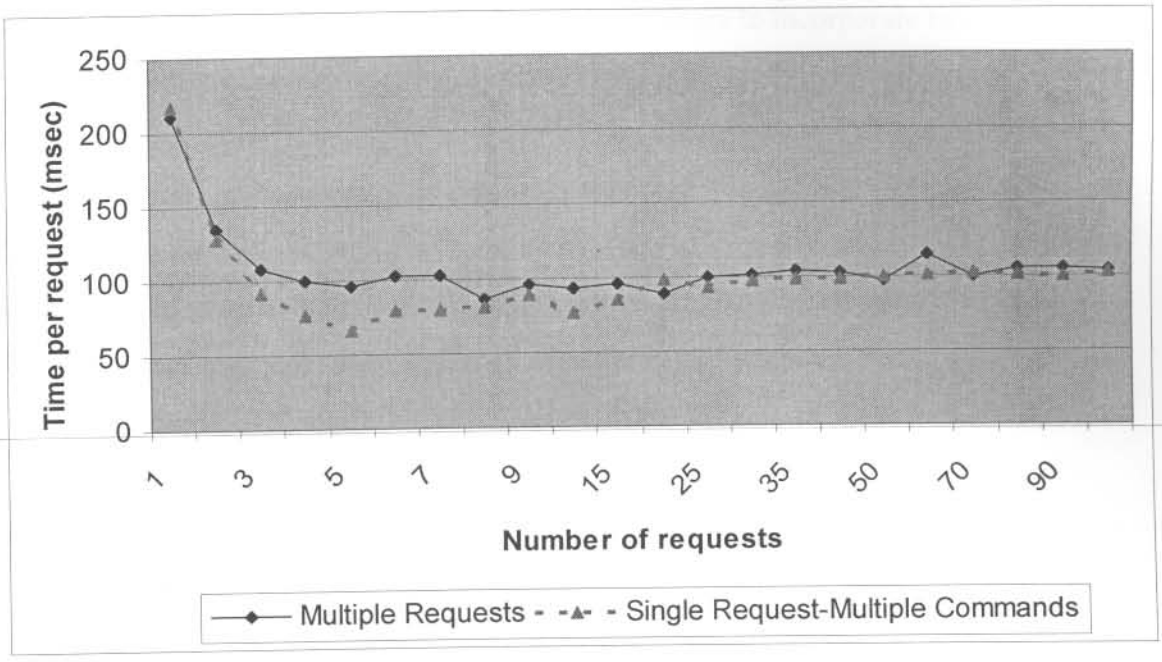

Figure 39: Graphical representation of time taken to process an XML command as the number of commands increase

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.