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.

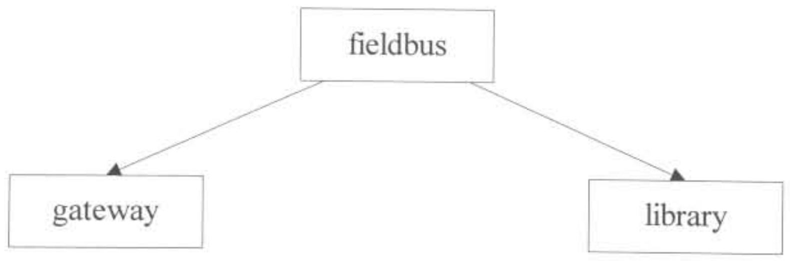


Figure 14: Fieldbus package diagram

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.

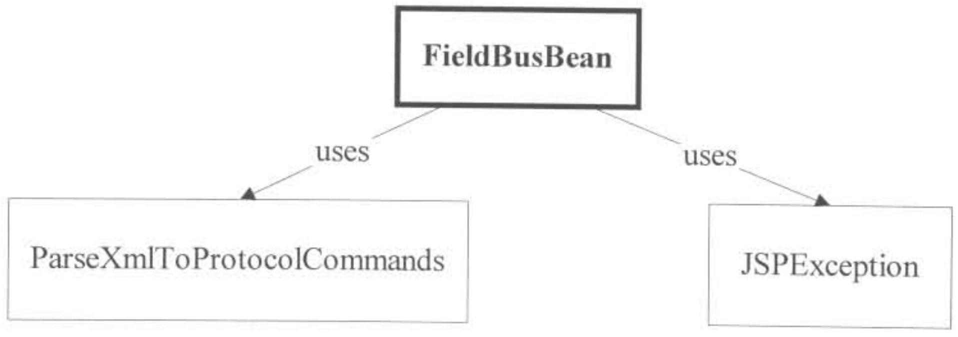


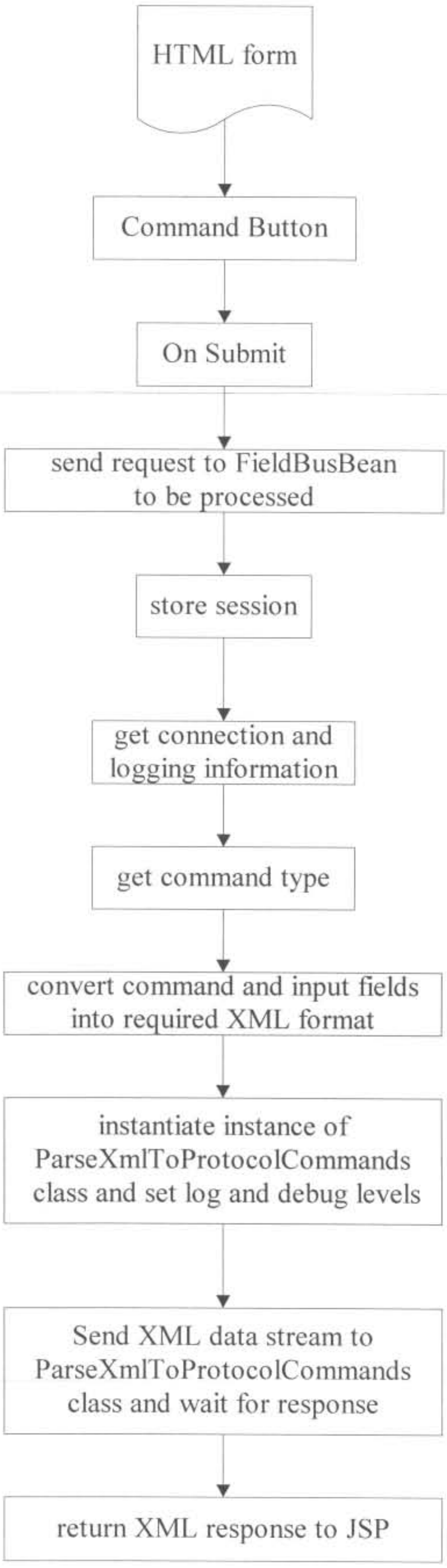
Figure 15: FieldBusBean class diagram

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.

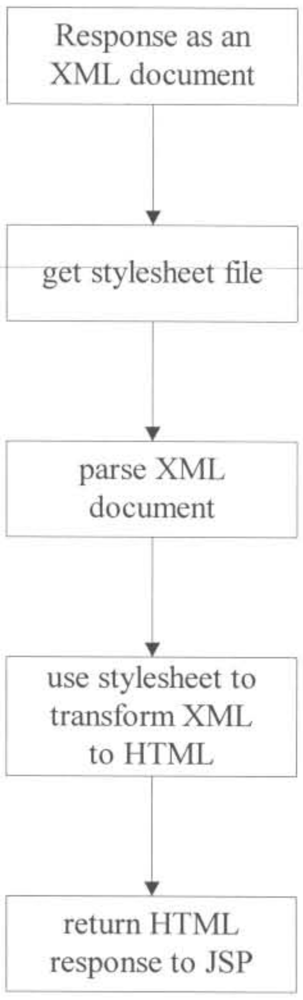
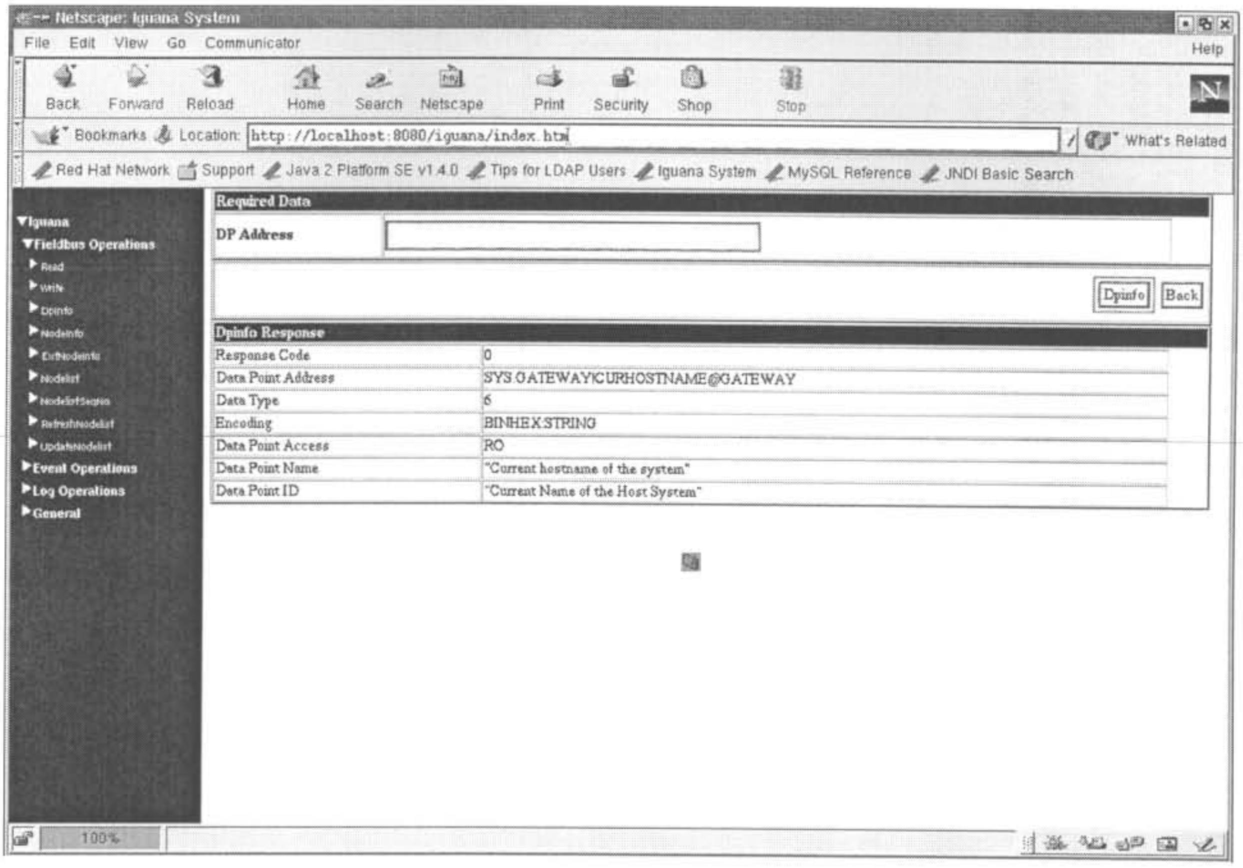

Response as an XML document
get stylesheet file
parse XML document
use stylesheet to transform XML to HTML
return HTML response to JSP

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.

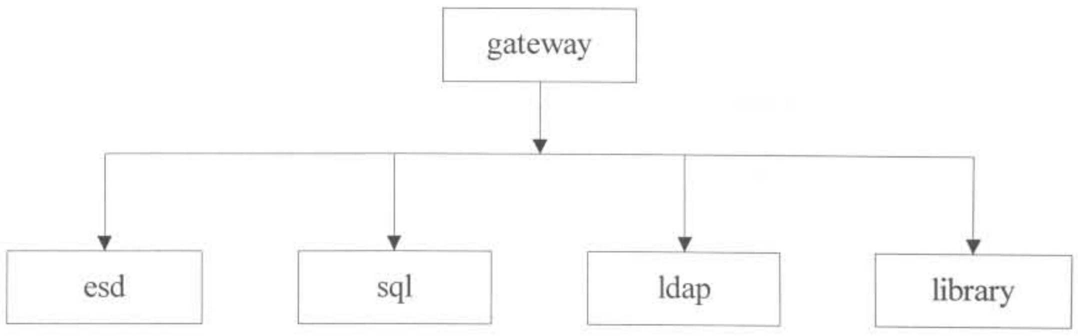


Figure 19: Gateway package diagram

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.

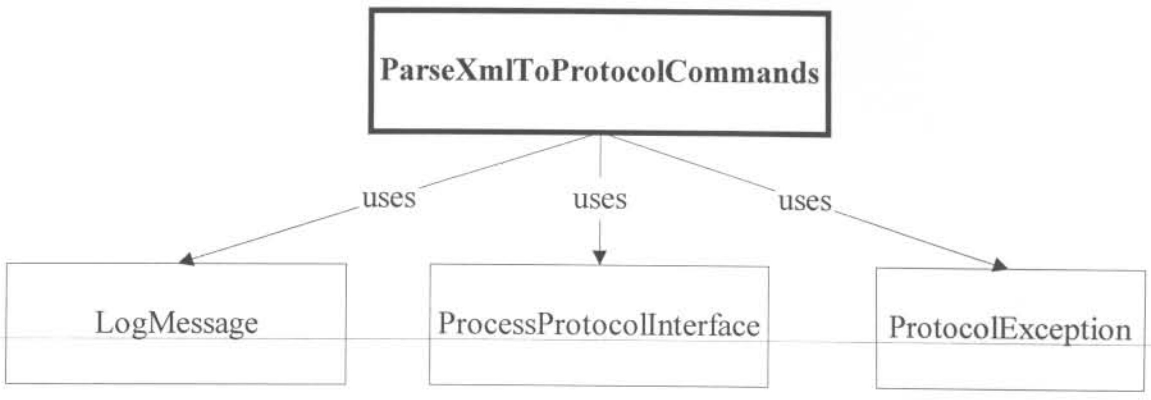
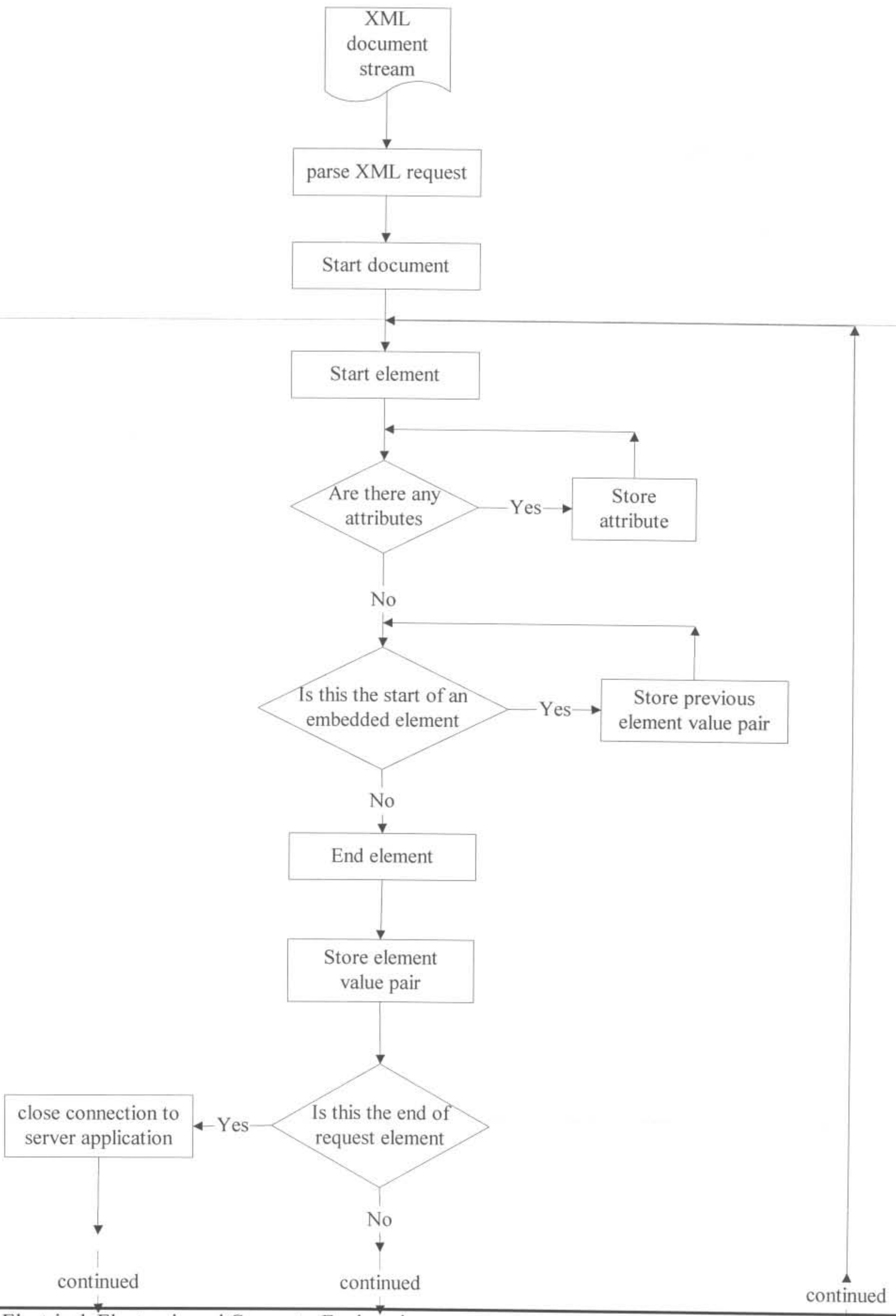


Figure 20: ParseXmlToProtocolCommands class diagram

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

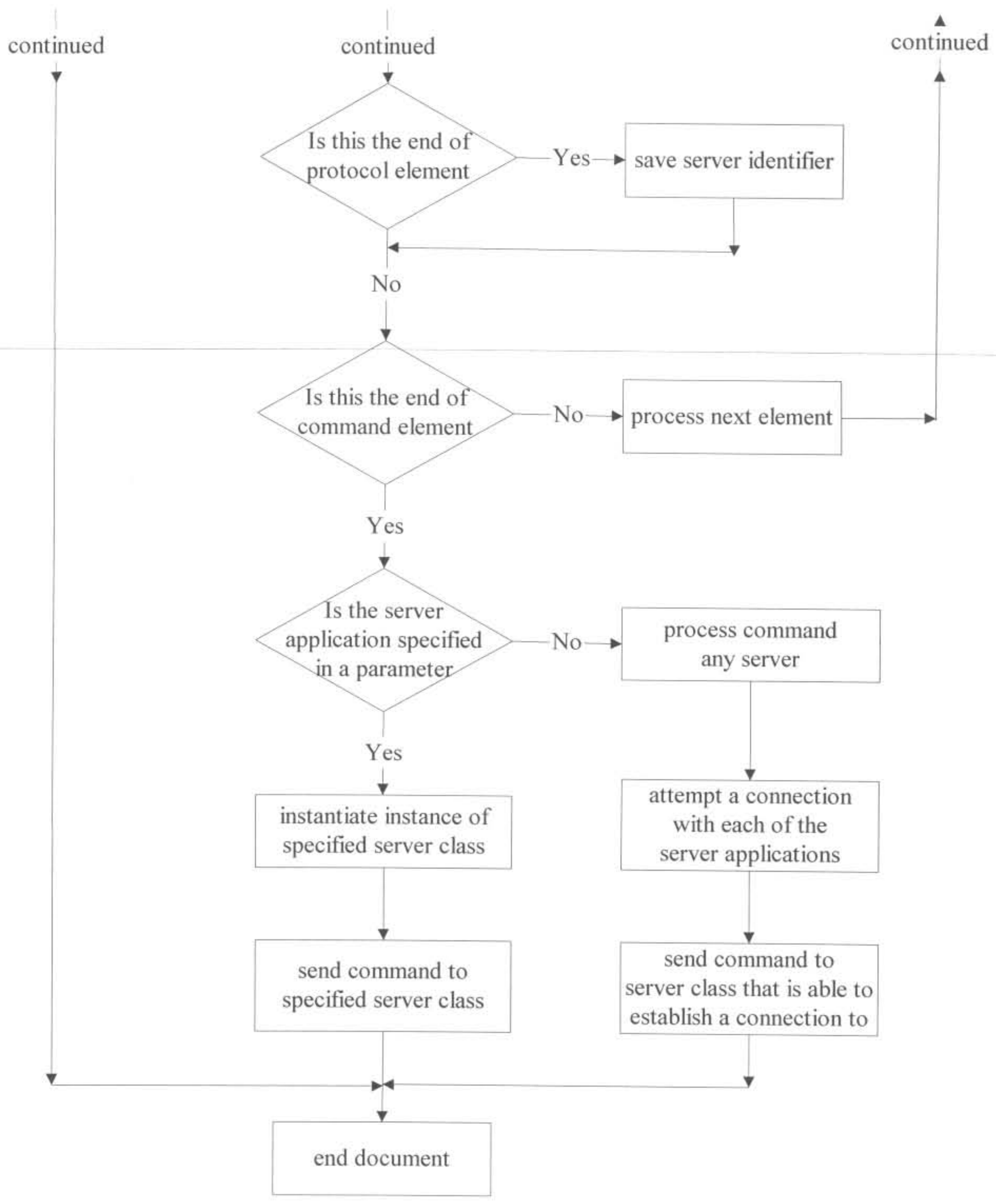

continued
continued
continued
Is this the end of protocol element
Yes
save server identifier
No
Is this the end of command element
No
process next element
Yes
Is the server application specified in a parameter
No
process command any server
Yes
instantiate instance of specified server class
attempt a 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.

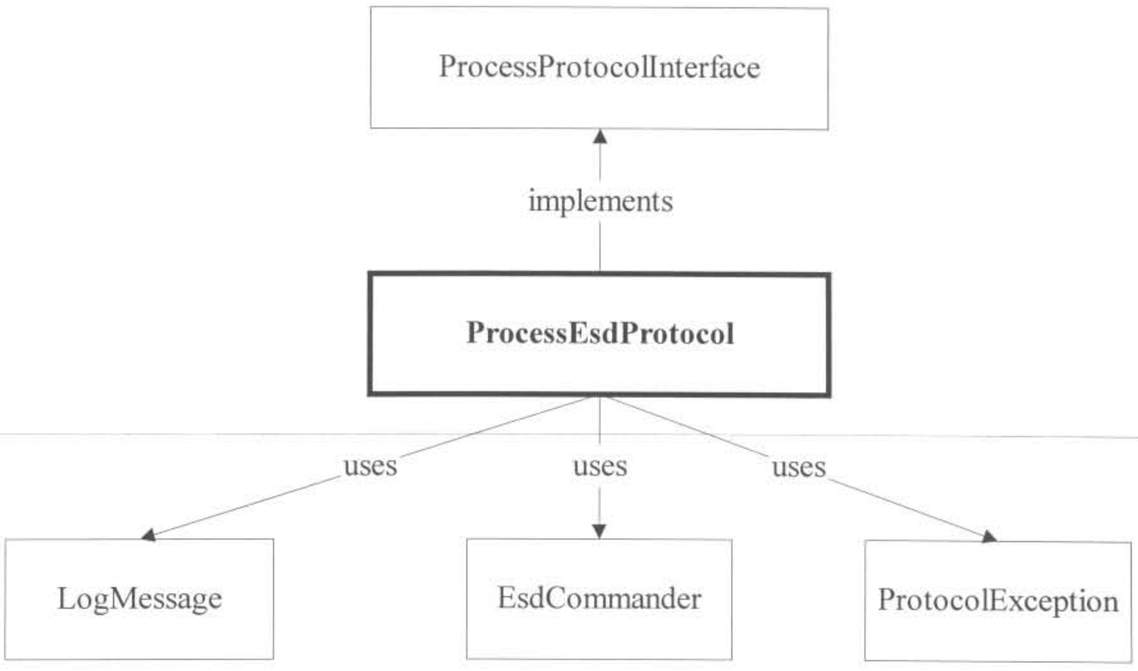


Figure 22: ProcessEsdProtocol class diagram

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

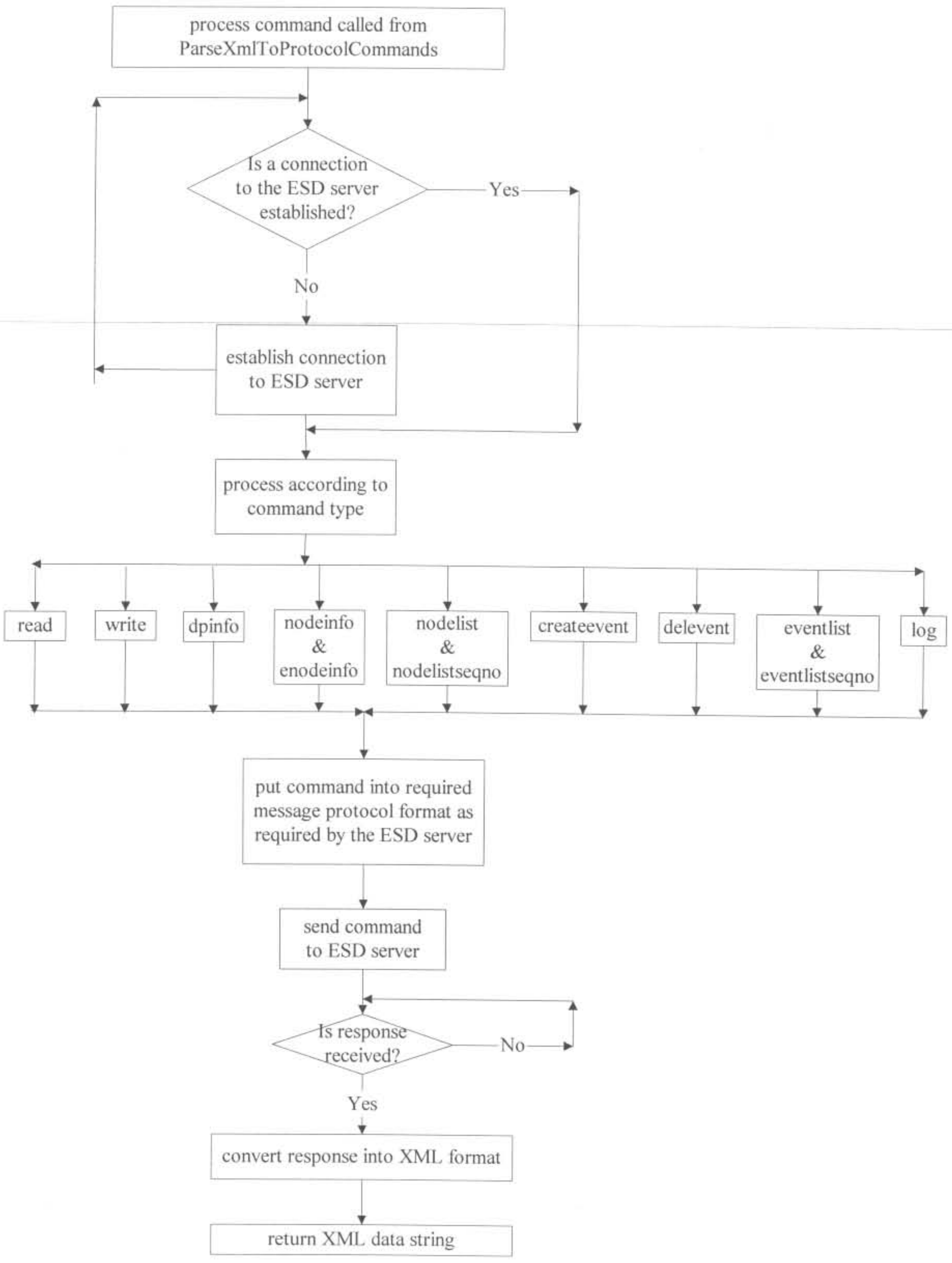

process command called from ParseXmlToProtocolCommands
Is a connection to the ESD server established?
Yes
No
establish connection to ESD server
process according to command type
read
write
dpinfo
nodeinfo & enodeinfo
nodelist & nodelistseqno
createevent
delevent
eventlist & eventlistseqno
log
put command 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.

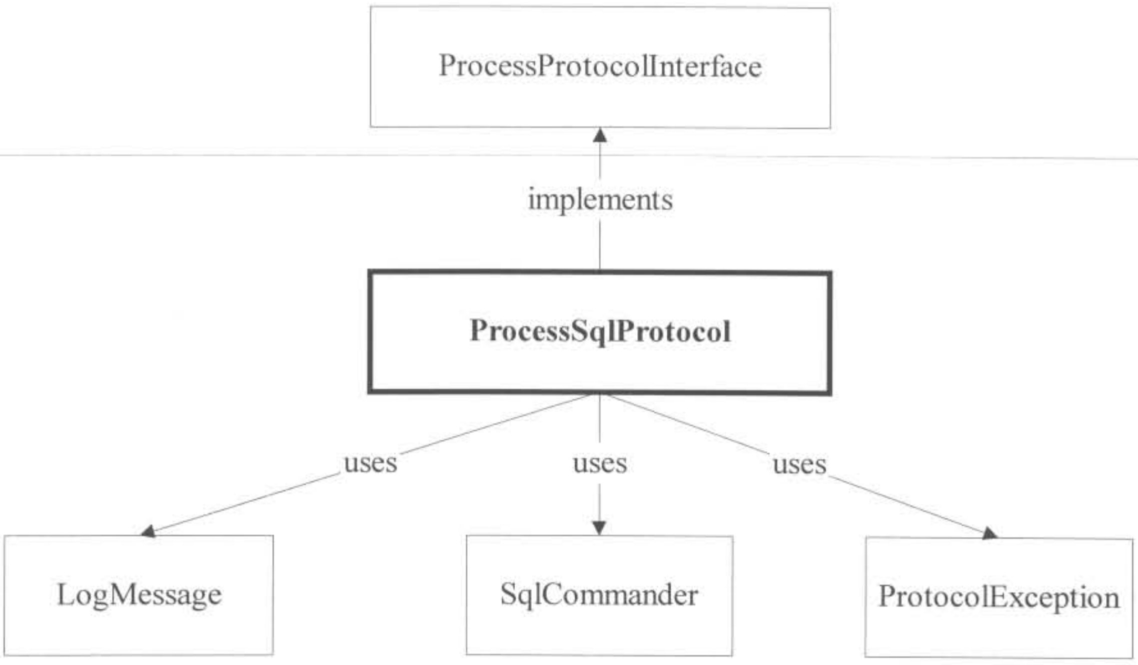


Figure 24: ProcessSqlProtocol class diagram

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

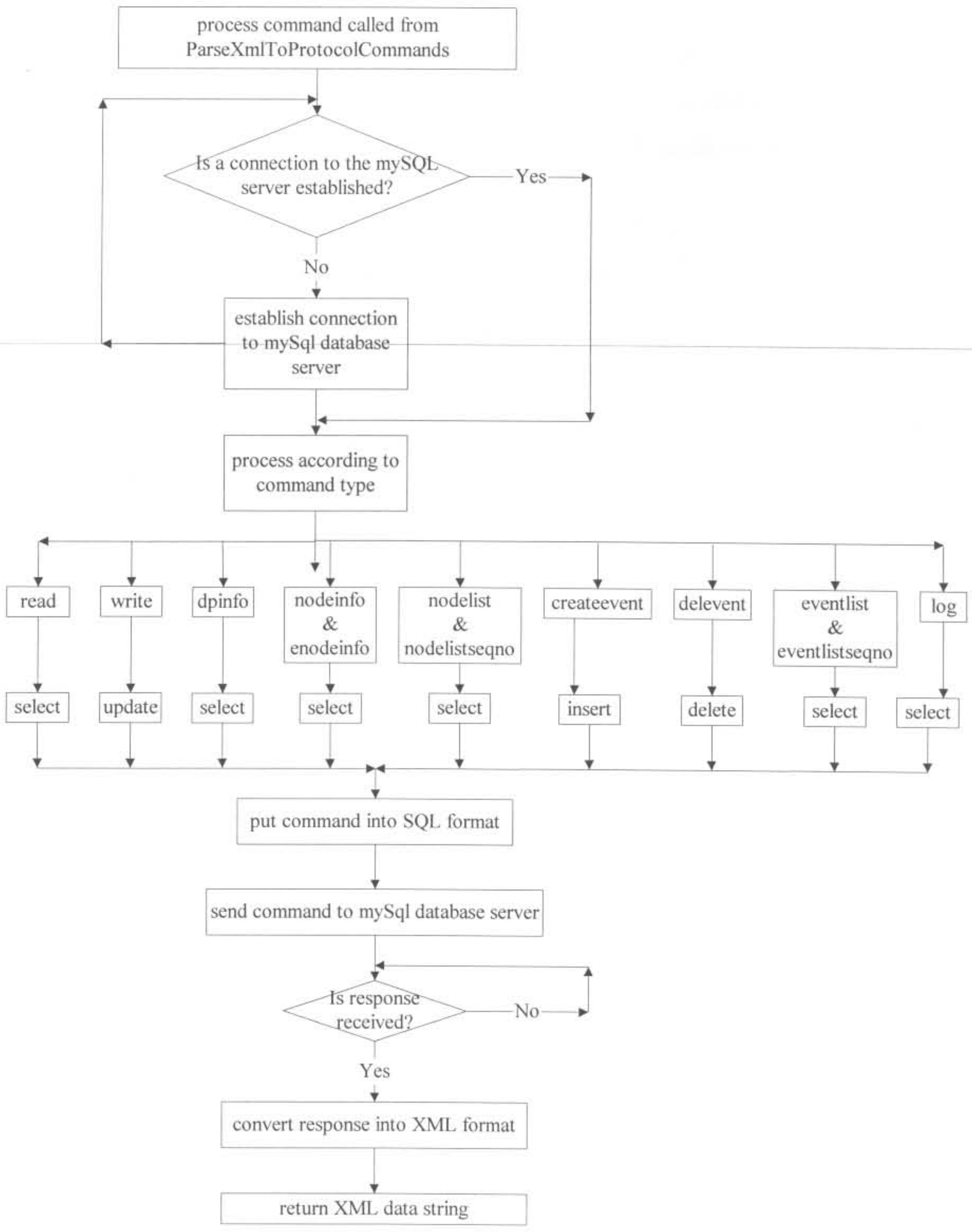

process command called from ParseXmlToProtocolCommands
Is a connection to the mySQL server established?
Yes
No
establish connection to mySql database server
process according to command type
read
write
dpinfo
nodeinfo & enodeinfo
nodelist & nodelistseqno
createevent
delevent
eventlist & eventlistseqno
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.

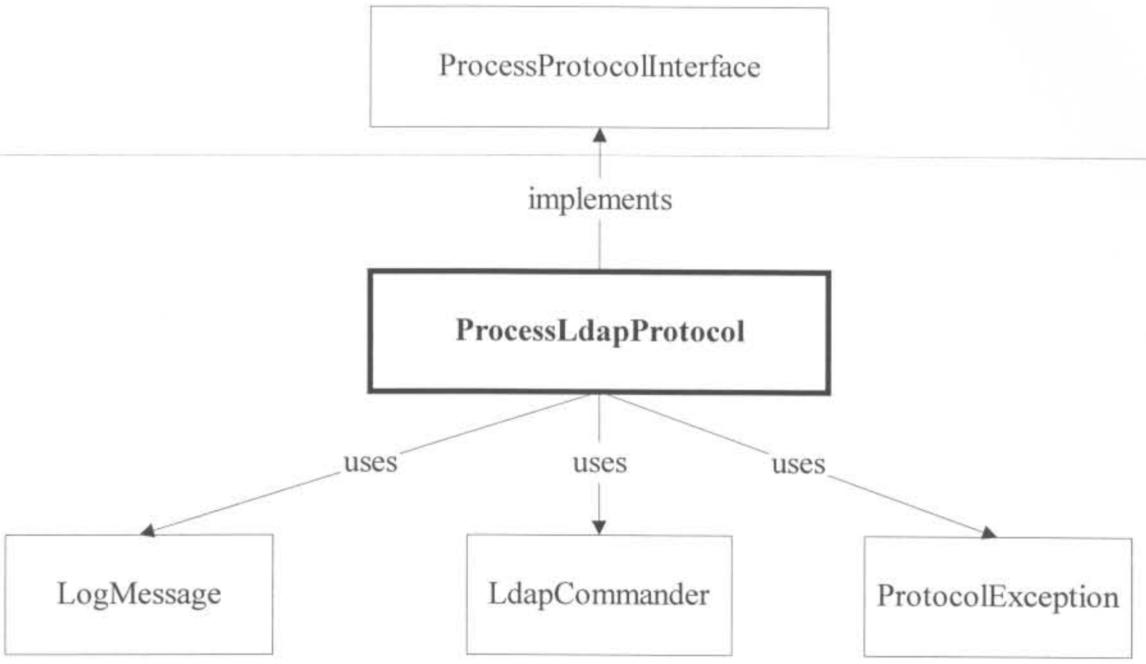


Figure 26: ProcessLdapProtocol class diagram

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

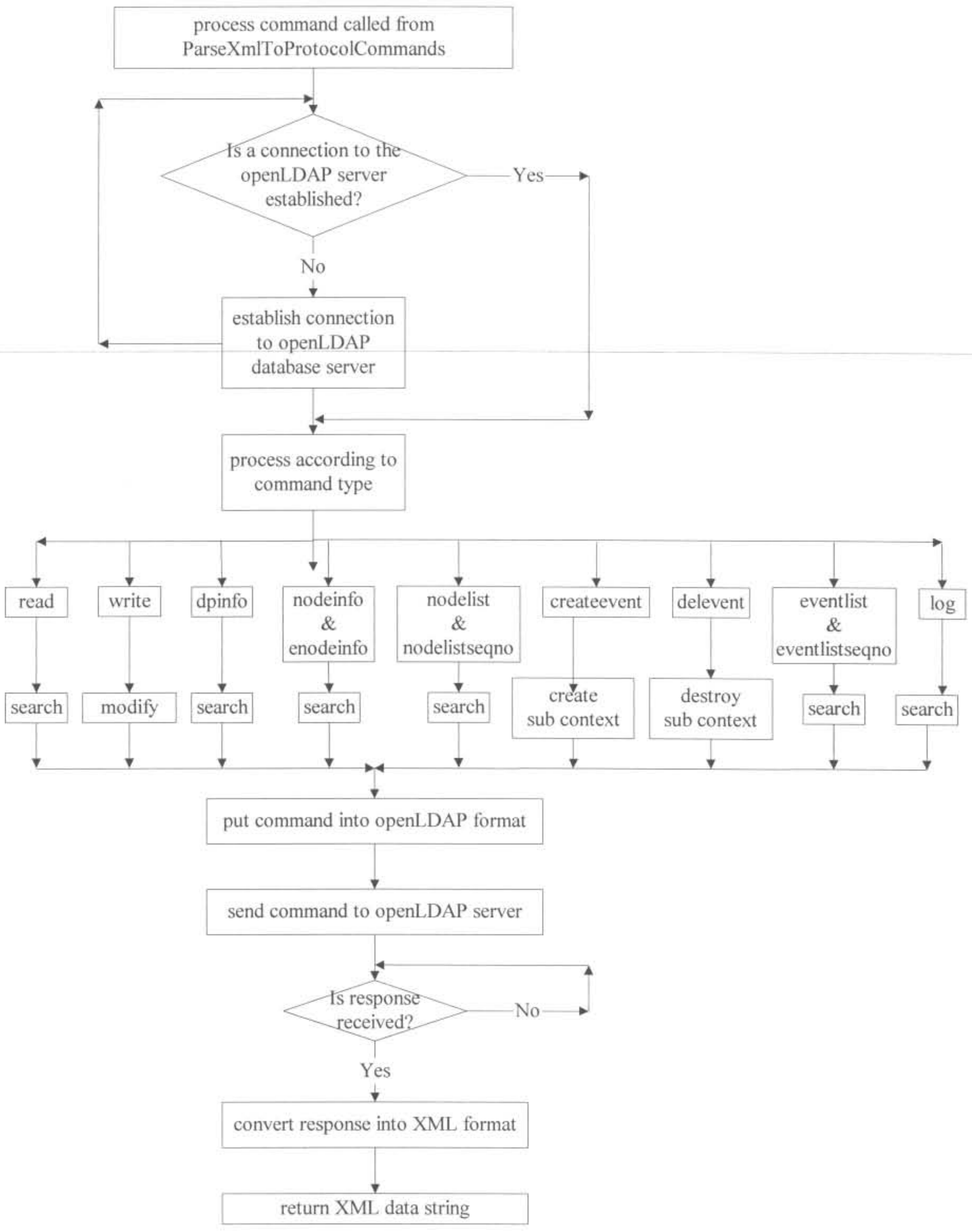

process command called from ParseXmlToProtocolCommands
Is a connection to the openLDAP server established?
Yes
No
establish connection to openLDAP database server
process according to command type
read
write
dpinfo
nodeinfo & enodeinfo
nodelist & nodelistseqno
createevent
delevent
eventlist & eventlistseqno
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.

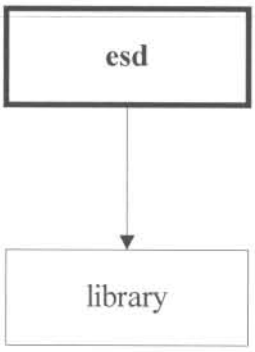


Figure 28: esd package diagram

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.

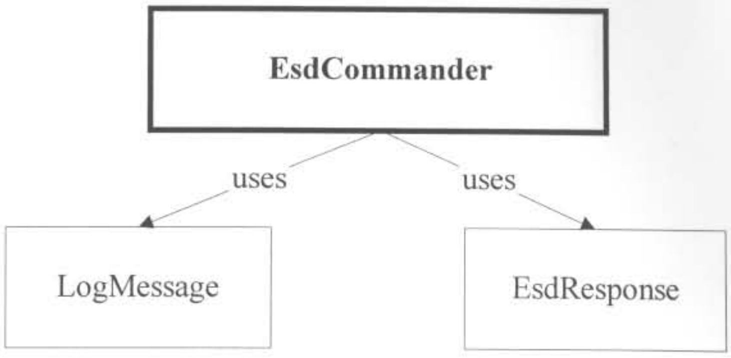


Figure 29: EsdCommander class package

7.2.3.2 sql package

The following figure shows the sql package diagram.

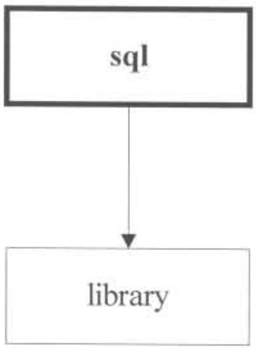


Figure 30: sql package diagram

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.

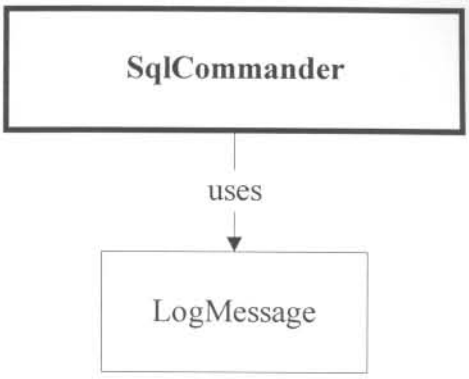


Figure 31: SqlCommander class diagram

7.2.3.3 ldap package

The following figure shows the ldap package diagram.

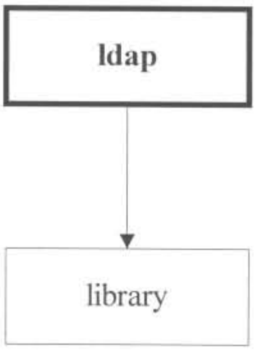


Figure 32: ldap package diagram

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.

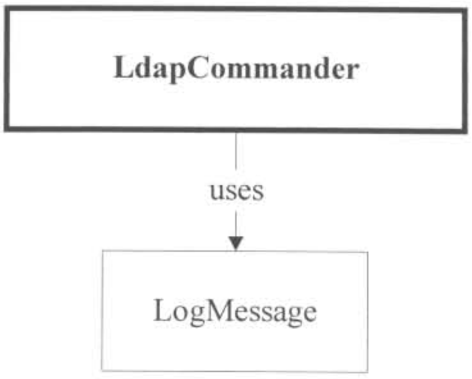


Figure 33: LdapCommander class diagram

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

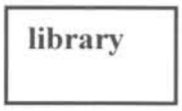


Figure 34: library package diagram

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.

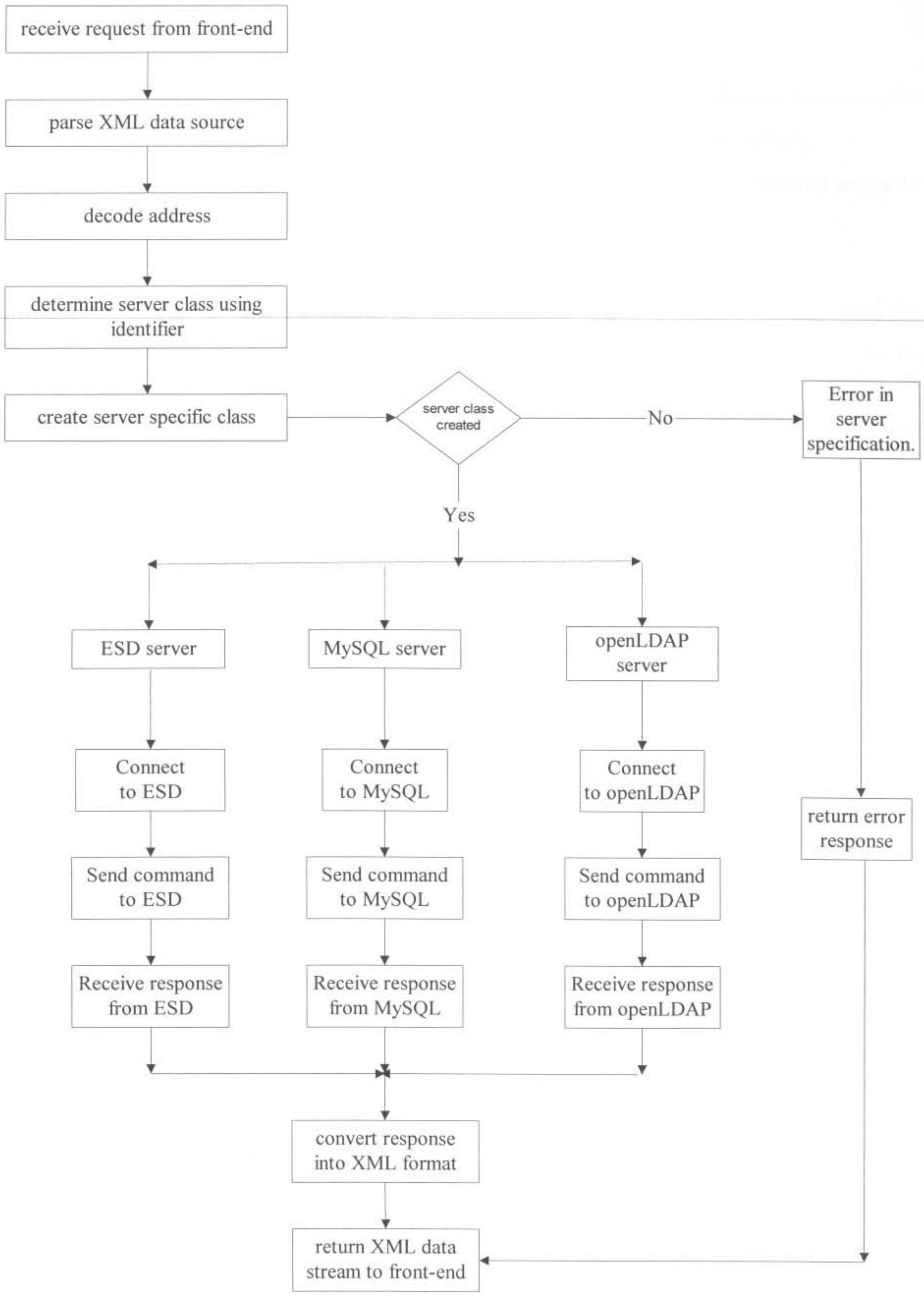

receive request from front-end
parse XML data source
decode address
determine server class using identifier
create server specific class
server class created
No
Error in server specification.
Yes
ESD server
MySQL server
openLDAP server
Connect to ESD
Connect to MySQL
Connect to openLDAP
Send command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