Information Collaboration on Mobile Networks

Fu Ting Chan

Email: ftc97@ic.ac.uk

Final Report

Course: Supervisor: Jeff Magee
Second Marker: Sophia Drossopoulou

MEng Computing IV
Department of Computing
Imperial College of Science, Technology and Medicine
180 Queens Gate, London SW7 2BZ, UK

June 2001

Source Code and User Manual Location: http://www.doc.ic.ac.uk/~ftc97/project
This project is concerned with the development of a framework that facilitates development of platform independent applications on mobile devices. The implementation consist of two distinct parts:

1. Development of a toolkit that allows communications between heterogeneous devices using the WAP Binary XML documents standard.
2. Development of a dynamically configurable server that accepts these Binary XML documents and initiate a chain of actions based on the content of the received documents. The server is capable of performing operations such as executing database queries, conditional branching based on input parameters and sending off a Binary XML document defined in a template.

The following diagram illustrates a high level abstract view of the above components and how they can form a framework to allow platform independent mobile application development. The binary XML toolkit facilitates XML based communication across heterogeneous platforms whilst the binary XML server compliments the framework by allowing dynamically configuration of process definitions and provides an interface for wireless device to access globally available information from a database.
Acknowledgements

I would like to thank the following people who have helped in various ways throughout this project.

My supervisor Jeff Magee for his time, guidance and allowing the original idea of the project to be pursued, and for the many interesting discussions that I have benefited from and always giving me the support and the “go ahead” when some new idea I’ve had were being discussed, allowing me to continually elaborate on the development of this toolkit.

My second marker Dr Sophia Drossopoulou for her insights on the future application of the Java API on heterogeneous mobile platforms.

My personal tutor Dr Iain Philips for his support over the initial idea and suggesting to me some possible areas of investigation of this project.

All of my colleagues, for their moral support and interest about this project. Some of the discussions has provided the motivation of new ideas and extra functionalities in the toolkit. Also thanks to those who have helped in testing the interface of the server configurations tool and for their help advice in possible improvements.
## Contents

CONTENTS ........................................................................................................................................... 4

1 INTRODUCTION .................................................................................................................................. 6
   1.1 THE PRESENT PICTURE .................................................................................................................. 6
   1.2 WHAT’S MISSING? A SCENARIO ................................................................................................... 6
   1.3 MY CONTRIBUTION ................................................................................................................... 7
   1.4 REPORT STRUCTURE .................................................................................................................. 9

2 BACKGROUND RESEARCH ........................................................................................................... 10
   2.1 XML TECHNOLOGIES .................................................................................................................. 10
   2.2 APPLICATIONS OF XML .......................................................................................................... 16
   2.3 DEVELOPMENT TOOLS AND APIs .......................................................................................... 17
   2.4 WIRELESS CONNECTIVITY ........................................................................................................ 19
   2.5 DATABASE SERVER ................................................................................................................... 20
   2.6 CURRENTLY AVAILABLE PRODUCTS ...................................................................................... 21

3 SPECIFICATION ............................................................................................................................... 24
   3.1 MOBILE COMMUNICATION TOOLKIT ........................................................................................ 24
   3.2 DATABASE INTERFACING SERVER ............................................................................................ 28
   3.3 USE CASES .................................................................................................................................. 32
   3.4 SYSTEM MODELS ...................................................................................................................... 40

4 PART 1 - DESIGN OF MOBILE COMMUNICATION TOOLKIT .................................................. 42
   4.1 DESIGN DECISIONS .................................................................................................................... 42
   4.2 WAP BINARY XML ENCODING BASICS .................................................................................. 43
   4.3 THE VISITOR PATTERN ............................................................................................................ 45
   4.4 BINARY XML ENCODER DESIGN CONSIDERATIONS .............................................................. 46
   4.5 ENCODER DESIGN .................................................................................................................... 49
   4.6 XML FRAMEWORK DESIGN .................................................................................................... 51
   4.7 BINARY XML PARSER DESIGN ............................................................................................... 55
   4.8 XML DOCUMENT SERIALISATION ............................................................................................ 56
   4.9 PLATFORM CONSIDERATIONS .................................................................................................. 56

5 PART 1 - IMPLEMENTATION OF MOBILE COMMUNICATION TOOLKIT ....................... 57
   5.1 IMPLEMENTATION OF DOM TREE .......................................................................................... 57
   5.2 IMPLEMENTATION OF ENCODER ............................................................................................. 57
   5.3 IMPLEMENTATION OF PARSER ............................................................................................... 59
   5.4 ERROR HANDLING AND DEBUG FUNCTIONS ........................................................................... 60
   5.5 ENCRYPTION AND COMPRESSION ......................................................................................... 60
   5.6 HASHCODE()FUNCTION .......................................................................................................... 60
   5.7 IMPLEMENTATION CHALLENGES .............................................................................................. 61

6 PART 2 - DESIGN OF XML SERVER ............................................................................................... 63
   6.1 DESIGN DECISIONS .................................................................................................................... 63
   6.2 USE OF COMMUNICATION TOOLKIT ....................................................................................... 63
   6.3 DESIGN OF SERVER ................................................................................................................... 65
   6.4 DESIGN OF CONFIGURATION TOOL ........................................................................................ 70
   6.5 XML CONFIGURATION DEFINITION ...................................................................................... 75

7 PART 2 - IMPLEMENTATION OF XML SERVER ........................................................................... 78
   7.1 SERVER IMPLEMENTATION ....................................................................................................... 78
7.2 Configuration Tool Implementation ................................................................. 82

8 Testing ................................................................................................................. 95
  8.1 Functional Testing ............................................................................................ 95
  8.2 Stress Testing .................................................................................................. 100
  8.3 WBXML Protocol Efficiency ............................................................................ 103

9 Evaluation .......................................................................................................... 104
  9.1 Achievements .................................................................................................. 104
  9.2 What It Can Do .............................................................................................. 106
  9.3 Limitations ..................................................................................................... 107
  9.4 Conclusion ..................................................................................................... 107
  9.5 Further Development Possibilities ................................................................. 108

10 Bibliography ..................................................................................................... 110

11 Appendix .......................................................................................................... 112
  11.1 Appendix 1 .................................................................................................. 112
  11.2 Appendix 2 .................................................................................................. 113
  11.3 Appendix 3 .................................................................................................. 114
  11.4 Appendix 4 .................................................................................................. 115
  11.5 Appendix 5 .................................................................................................. 117
  11.6 Appendix 6 .................................................................................................. 119
  11.7 Appendix 7 .................................................................................................. 120
  11.8 Appendix 8 .................................................................................................. 122
  11.9 Appendix 9 .................................................................................................. 123
1 Introduction

1.1 The Present Picture

Advances in Wireless Connection Technologies such as WAP and the development of the 3rd Generation Mobile Networks is promising to open up opportunities in fields where high mobility, high connectivity applications are required. Opportunities in this market “niche” is opening up, scenarios of current applications in wireless technologies range from a financial derivative analyst reading up on present market dealings on his way to work to someone activating his microwave using on a mobile phone whilst he’s walking home.

Leading-edge portable microcomputers equipped with the latest wireless connection technologies are an increasingly popular platform for the implementations of such mobile applications. As a result more and more time critical information are stored, and transmitted using these distributed mobile devices.

1.2 What’s Missing? A Scenario.

To illustrate what is missing from the current picture, let’s consider the following scenario:

I am a professional mobile application developer, and am currently working on a project to develop a calendaring application that allows users to enter their calendar events on their mobile device and to share these calendar events with another user. I have already designed and implemented two separate versions of the calendaring application that runs on Palm and Windows CE devices, as this is what I was told the users of the system would be using. Wireless communication allows me to send and receive individual bits between these devices but I must now find a way to be able to transmit the calendaring events across.

At the present there are three main ways to transmit information from one mobile device to another:

1. Device Information Synchronisation
2. WAP WML (Wireless Markup Language)
3. Proprietary Protocols

Mobile device manufacturers usually bundle with their device a software that allows users to synchronise information from the mobile device with a PC or a mobile device with another mobile device of the same type. The present process of mobile device synchronisation relies heavily on application and platform specific protocols that are proprietary. If my users have mobile devices that run on different platforms, I would have trouble trying to use this method to communicate between these devices. Also, since I have no control over the synchronisation process, if I cannot add any extra information to be transmitted in my calendaring system.

I can use the WML standard that was published by the WWW consortium. I will need to set up a WML server that can be accessed from mobile device WML browsers. The server would store all these calendaring events so that mobile devices that visit this server can access and exchange the information. WML provides
me with a platform and application independent way of mobile device communication, however it is indirect, information cannot be exchanged directly between two devices. Information is stored at the server site, we do not have much control over the WML browser, therefore mobile clients cannot store these information easily into a local database for offline viewing. WML is also highly geared toward user interface and presentation objects, so it will be inefficient to transmit a lot of non user interface data using WML documents. Another problem with WML is that it is built based on a request reply model, so there is no easy way to “push” some particular information to a client application, for example notifying a user of a newly added calendar event.

Given the shortcomings of the above two processes, I may end up designing my own protocol to communicate between my applications, as I need the control and efficiency over the way my data is being transmitted over the wireless link. This is highly undesirable in today’s world of standardisation and continuous integration, it would for example be difficult for me later on to allow my application to communicate with an external system that runs on a different protocol as the one I designed.

It is clear that the main challenge in the current field of mobile application development is to overcome the inherent platform, application and protocol heterogeneity that is constraining on the portability of information stored on mobile devices. The key problem is that we have portable device with non-portable information. The emerging wireless connectivity technology gives us an opportunity to compensate for this but we are missing a framework for developing wireless application communications that would give us control, efficiency and compatibility.

1.3 My Contribution

The aim of this project is to design and implement of a framework that allows wireless applications to communicate with each other effectively. Mobile application developers using this framework should have the full control over the content of the communication and should not have to worry about the underlying device platform or communication network that is involved. The framework should have a clear separation between data and presentation and does not confuse user interface components with data components. The framework should also support different kinds of network access models, so that a variety of mobile applications could be developed using it. The framework should also promote integration and compatibility, so that applications using this framework are extendable and can communicate with external systems easily.

Conceptually, the following two diagrams shows what the framework aims to achieve in this project. The Figure1 shows the current ‘disconnected’ picture of the current state of the art in mobile application development. Figure 2 shows what this project aim to achieve.
Figure 1: Current Picture

Figure 1 shows that at the present we cannot achieve application independent communication between different platforms. The blue arrows indicates the use to specific application protocols that works on usually the same platform. We can however use WML to indirectly transmit data between platform, but it’s shortcomings was discussed in section 112 for this purpose.

Figure 2: Aim of this project (same key as figure 1)

The primary goal of this project is to design and implement a communication toolkit (yellow shaded circle in figure 2) that allows direct, application and platform independent communication between wireless devices. The toolkit would provide a common interface for application programmers to communicate between different platforms whilst being able to control freely the application that is being constructed.

The secondary aim of this project is to enhance the process of mobile application development by implementing a dynamically configurable server to allow wireless applications to have database querying capability using this communication toolkit. This would greatly help developers because a lot of applications
running on mobile device would need to interface to a database at some point, for example it may need to lookup records within a database to authenticate a user or it may need to insert a new record to the database of calendaring events. This server can automate this part of the process, by allowing application developers to define templates of input information that this server is allowed to receive and map this to a sequence of operations that needs to be done with the input. The following diagram shows what this second part of the project is about.

![Figure 3: Database interfacing server for mobile clients](image)

The two shaded circles represents the toolkit that has been developed in part one of this project, the key is that the same toolkit is used for both the development of the client running on the wireless device and the server accepting this input information, this is made possible because one of the important requirement that I have mentioned in part one was platform independence. Within the server, we also have a blue circle which defines the templates of information that this server is allowed to accept and the sequence of actions associated with this information. By allowing the templates to be defined dynamically, we can alter the behaviour of the server easily. The figure shows a particular scenario where a wireless application performs a database operation, it first sends a request to the server using the toolkit (1), the server receives this information using the toolkit, it then looks up the set of template definitions it has (2) and perform the appropriate database operations required (3, 4). The server then constructs a reply based on the template mappings (5) and sends it back to the wireless client using the toolkit(6).

### 1.4 Report Structure

The remainder of the report is structured as follows. Chapter 2 gives an overview of the relevant state of the art in wireless application development and the technologies that I may need to use to build my system. Chapter 3 includes details of how I have constructed the specification of the system. Chapter 4 summaries how I approached in designing the communication toolkit and the database-interfacing server.
2 Background Research

In this section, I will investigate into the current state of the art of the new field of mobile application development. First I would look in to various XML technologies, which is a promising way of encoding, transmitting and storing data at the same time maintaining application and platform independence. Then I would provide an overview of the development environments that are available for mobile application. Lastly I would summarise the key strengths and weakness of a few currently toolkits that allows mobile application communications.

2.1 XML Technologies

XML technologies provide a platform independent language for interchanging data. Using XML as the protocol for communication in my system would ensure that my system is both extensible and capable of operating on different platform independently. In this subsection I will present the findings of my research into XML, XML protocols, the frameworks available to parse XML and a few parsers available for parsing XML.

More information about XML technologies can be found at http://www.w3c.org. A good book that is worth reading is a book by Brett Mclaughlin called “Java and XML”, it gives a very good introduction to XML technologies and discusses how the use of Java with XML together can produce truly platform and data portable solutions.

2.1.1 XML Framework

XML which stands for Extendible Markup Language is a definition of a metadata-type framework that provide a uniform structure for data to be stored, managed and transmitted. This allows data to be integrated across multiple heterogeneous systems. XML 1.0 is a completed W3C recommendation.

The power of XML lies in the fact that the tag set of the language is not defined within the specification but is rather influenced by the context we associate the XML document. By defining neither the tags nor the grammar, XML is completely extensible and flexible in applications. This strong point is the main reason why I am looking into the possibility of adopting the technology in my system

2.1.1.1 DTD

DTD which stands for document type definition, is a part of the XML specification. It establishes a set of markup constraints for an XML document to define the way the document is to be constructed. The constraints that a DTD defines for an XML document includes the allowed element and attributes in that document, possibly the acceptable attribute values for each element, the nesting and occurrences of each element.

DTD has a structure completely unlike XML this is due to the fact that DTD was originated from a much older specification of SGML DTD could be used in our system to constrain the XML documents that is used to represent mobile information in our system, if an incoming XML document violates some of the constraints set by it’s DTD’s we can report such error or simply discard the incoming request.
2.1.1.2 XML Schema

XML Schema is designed to replace and amplify DTD. It adopts a much more XML centric means to constrain XML documents and resolves some of the weaknesses of DTD. A XML Schema constraints it’s corresponding XML document in a very similar way as DTD, by defining the elements and attributes that the XML document is allowed to have, and also the nesting and occurrences of the elements. However, unlike DTD, XML schemas are actually XML documents that are both well-formed and valid, This allows parsers and other XML-aware applications to handle XML Schema documents in a fashion similar to other XML documents, the parsers would not have to adopt special techniques for handling DTD documents.

A major drawback in using XML Schema in my system would be the fact that XML Schema is both a new and young specification, it’s specification is incomplete and it’s implementation is subject to change. This is because this is still a draft specification from the W3C. As a result many parsers do not support XML Schema, or support only a portion of the schema.

2.1.1.3 XSL and XSLT

XSL which stands for Extensible Stylesheet Language transform and translates XML data from one XML format into another. This would be particularly useful in my system in the transforming of data from XML to HTML, WML and PDA protocol if later I decide to allow my server to interface clients to different types. It is an elegant solution that would promote reusability in the data access layer. XSLT (Extensible Stylesheet Language Transformation) is the process of an XSL Stylesheet and an XML document merging together resulting in a formatted XML document.

2.1.1.4 XQL

XQL is a query language to allow XML document formats to easily represent database queries. The standard is becoming more and more popular, and it’s usefulness is likely to make it the de facto method for specifying access to data stored in a database from an XML document. The results to XQL is returned as standard XML documents, defined using XQL specific tags. It is possible to use XQL as a possible interface to database queries from mobile applications, the problem is however, there is not at the moment an easy way to encode these XQL using a mobile device, and the inherit nature of string tags and attributes of XML means that the return result might be very large, it will therefore be difficult if connection speed is a concern for wireless devices. Another problem with XQL is that it is not yet formally adopted by the W3C standards committee.

2.1.2 XML Protocols

There are a large amount of protocols available for transmitting data using XML. Each has their specific applications, advantages and disadvantages. I looked at various XML protocols that are widely used and could potentially be used in as the communication protocol to use in my toolkit. Full definitions of each protocol could be found at W3C’s website at http://www.w3c.org.

2.1.2.1 XML-RPC

XML-RPC is a remote procedure calling protocol that works over the internet using XML objects to represent the calls. Procedure parameters can be scalars, numbers, strings, dates and can also be complex records such as list structures. An XML-RPC message is an HTTP-POST request, after the
procedure has been executed on the RPC server, the value it returns is also formatted in XML. The protocol follows a similar request-reply model that is required when retrieving remote information for mobile devices. However, if this protocol is used in my system, a lot of redundant information such as headers, method call information and payload formatting information would be included in the communication between the mobile devices. Since the protocol is not very compact, it is not very suitable for wireless mobile communication where communication bandwidth is limited. Another problem of using XML-RPC is that it relies on the fact that we are calling a remote procedure to perform specific actions, whilst this may be useful for my database-interfacing server, it will not be suitable for normal data transmission between devices. The XML-RPC specification can be found at http://www.xmlrpc.com/spec

2.1.2.2 SOAP

SOAP which stands for Simple Object Access Protocol provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. SOAP defines a simple mechanism for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules. SOAP is a Net standard certified by the World Wide Web Consortium (W3C) Internet standards group. SOAP supports XML schema, and is based on a request reply mechanism similar of that found in HTTP. SOAP defines a serialization mechanism that can be used to exchange instances of application-defined datatypes. An interesting attribute that SOAP has is that SOAP can bind with the HTTP protocol to allow a SOAP message to be carried in HTTP messages either with or without the HTTP Extension Framework.

This attribute can be useful in my system as the mobile client can act as a web client performing a POST request with a SOAP object representing the information that it needs, this request can then be passed to the appropriate processing thread from the web server. So in essence the database-interfacing server can act a thread sitting behind a web-server that conducting database operations based on request. This means that there would not be a need for a separate server listening on a separate socket from the web server waiting for a mobile client to connect. However a major drawback from using this method is the fact that HTTP is a stateless protocol, once a request has been served, the thread that handles the request also stop. For example if we would like to store a session variable indicating whether or not the client has been authenticate before performing operations defined by a request, then we would have to effectively authenticate the user on each individual request, this would be a high communication cost and processing cost.

Another reason why SOAP will not be suitable for our application is that SOAP is not as compact as some of the other protocols we will examine. The SOAP specification could be found at http://www.w3.org/TR/SOAP/.

2.1.2.3 WDDX

Web Distributed Data Exchange was designed to facilitate the exchange of complex data within applications and between application environments. WDDX packets are XML documents that represents data structures that can be exchanged across heterogeneous systems. WDDX DTD can be
used to validate WDDX. The main difference between WDDX and other protocols is that WDDX is a high level API that is built on top of DOM (Document Object Model), it provides the infrastructure for typing data and validating structures. Unlike SOAP, WDDX documents are not objects, so it is not suited as a backbone technology for distributed object applications running on the web.

The advantage of using WDDX in my system is the fact that there would be readily available high level parsers already available for use in a congenital programming such as Java. This means that I would not have to construct a parser myself for the WDDX documents using low level tools. Another consideration is that WDDX are designed to be more focused on data exchange than the previous two protocols, a WDDX packet has much less redundant information to do with procedure calling or object structure, this is compatible with the aim of our information synchronisation system. A WDDX packet however, is still relatively large for transmission over wireless network and there is no easy mean for compacting the packets. WDDX packet specification could be found at http://www.wddx.org/DistributedDataForWeb.htm.

### 2.1.2.4 SyncML

SyncML, the initiative founded in Feb 2000 by mobile technology industry leaders Ericsson, IBM, Lotus, Motorola, Nokia, Psion, Palm, Inc. and Starfish Software. The scope of the SyncML initiative is to define a synchronization standard for all kinds of mobile devices such as PDAs, portable PCs, Communicators, Pagers and Mobile Phones.

SyncML compliant products will be able to exchange information seamlessly across a wide range of operating platforms and communications technologies. SyncML remote synchronization enables users to quickly and smoothly update and synchronize any kind of application, such as e-mail, calendar and to-do-lists, wherever they are using their mobile mobiles. SyncML leverages Extensible Markup Language (XML). The white paper of SyncML can be found at [http://www.syncml.com](http://www.syncml.com). However the specification was not published at the time of writing this review during the early stages of this project.

### 2.1.2.5 WBXML

WBXML is a binary standard for the representation of XML documents, it is approved by the Wireless Application Protocol (WAP) Forum. The protocol is designed with the fact that the limited bandwidth and processing power of wireless devices in mind, the binary protocol is compact and allows effective use of XML data on narrowband communication channels A WBXML document has the following structure of the following:

1. **Version** – Version Number for the XML document
2. **Document Public Identified** - to identify the type of XML document
3. **Character set** - Describes the character set used
4. **String table** - Contains a sequence of strings that can be referenced in parts of the document.
5. **Document body** - Consists of tags, attributes and contents.
The use of WBXML protocol in my toolkit has the advantage binary format’s compactness, encoded documents can take considerably less space than their traditional textual versions, and require less processing to parse and use in software.

WBXML can be used in our system by simply defining tag and attribute token values for each language used. WBXML also contains representations for tags and attributes that do not have corresponding tokens and even a mechanism for extending the encoding on a document-specific basis.

However, using WBXML has its drawbacks too. Most obviously, all comments and document meta-information is discarded when it is encoded. In addition, even though the document structure is preserved, the layout of the original XML is lost, making it more difficult to use and edit for humans.

It can also be argued that WBXML is not more compact than the other XML standards which have been investigated above. Although XML is rather verbose in its textual form, it compresses very well with normal data compression algorithms. Having said that, it is still much better to use a binary XML standard such as WBXML then to compress a textual XML document and transferring it across network. This way we would not have to bind our application to any particular compression algorithm, and it will be computationally less heavy for our mobile device with limited processing resources.

### 2.1.3 XML Parsing and Manipulation

There are a variety of tools and models that are available to help the manipulation of XML documents. I have listed the technologies that I have researched below.

#### 2.1.3.1 SAX

SAX is the Simple API for XML. It provides an event-based framework for parsing XML data. The XML document is processed and broke down into usable parts. The frameworks operates by defining a content handler interface class with methods such as `startDocument()` and `startElement()`, customised behaviours can be added by extending the handler class and implementing the interface methods to perform specific operation given the existing of an element or attribute. A set of errors and warnings that can occur in XML parsing is also defined in the handler, so that developers can add code to ensure that the document is well formed and well defined. The SAX framework can be coupled with a variety of parsers including Sun’s Project X for parsing in JAVA, the Apache Software Foundation’s Xerces, Oracle’s XML Parser, and IBM’s XML4J.

Although the SAX framework provide a convenient interface for handling XML documents, it’s use can sometimes be cumbersome, as different document handing code for the same document must be added various methods in the interface and handling operations frequently need to reference a class variable to find out the result read in from other methods. Also SAX parses an XML document as the document is read in, therefore not the whole document is available at all times during the parsing process.
2.1.3.2 DOM

DOM (Document Object Model) is an API that allows manipulation of XML documents by parsing the whole XML document into a tree structure. Manipulation of the document is mapped to traversal in the tree, XML data are stored as nodes. DOM reads the entire XML document into memory before manipulation, therefore access to the document structure is very fast, and it is all in memory for the length of the existence of the DOM tree. Manipulation of the document is also easy, with nodes defined as XML elements, we can access the attributes and child elements easily in a hierarchical fashion.

![DOM Tree Representation of a HTML table](image)

The figure above shows a DOM tree of a table structure in HTML. The structure has a root node called `<TBODY>` which has two child called `<TR>` being the records of the table. We could use DOM to part the above XML document, the fields of the table could then be accessed by using the reference to the root node we have, so for example to obtain the contents of Dorian in the table, we may first obtain a reference to the rootnode TBODY ie:

```java
Node tBody = Document.getRootElement()
```

We then obtain the contents by traversing down the tree using the getChild() method on the root node giving the relevant position of each child. Ie:

```java
TBODY.getChild(1).getChild(1).getChild(0),
```

The advantage of DOM is that it is very fast at performing specific operations on the XML document because of the well-defined algorithms available to carry out tree traversals and search. DOM however has the disadvantage of taking more system resources, as every time an XML document is parsed, the whole document must be stored into a tree structure in memory.

2.1.4 XML parsers implementations

After careful examination of the present popular XML parsers implementations, I cannot find one that would operate platform independently on mobile devices. I found one implementation of the SAX framework on Palm from an individual who has posted his software in CVS (under de.trantor.wap), however the implementation of the encoding class does not work properly.

The closest to a working XML parser on mobile device would be a WML browser, but it is not generic parse for XML. Never the less, I studied a few commercial parser implementations which seems
interesting and would be useful in this project to get a feel of how XML is parsed and processed. These parsers although does provide multi language support including JAVA, but the limited API of the J2ME means that these parsers would not work under mobile devices equipped with J2ME. The parsers that I looked at includes:

2.1.4.1 Apache Xerces parser

The Apache Xerces Parser is a popular open source parser freely available at http://xml.apache.org, it’s implementation is provided in a variety of languages including Java, Perl, C++ and COM. It implements the W3C XML and DOM (Level 1 and 2) standards as well as the SAX standard.

2.1.4.2 IBM XML 4J

Another parser that provides support in both DOM and SAX in a variety of languages. The parser is consider to be one of the most robust XML parser available. More info at http://alphaworks.ibm.com/tech/xml4j

2.1.4.3 Sun Microsystems Project X

Project X, contains a suite of utility for XML manipulation in Java. It’s XML parser called JAXP. JAXP is Sun’s Java API for XML Parsing. It conforms to the SAX and DOM specification. As the name describe, the API allows manipulation of XML documents using the JAVA programming language. JAXP provides cohesiveness for Java developers to the SAX and DOM APIs through a standard pluggability layer, therefore using JAXP a Java developer is not tied to any particular implementation of parsing XML documents whilst maintain a degree of control and access of the document. More info at http://java.sun.com/products/xml.

2.2 Applications of XML

We will briefly look at how XML could be applied in our system, and the benefits or problems that we might have when using XML in each of the following areas.

2.2.1 XML for Presentation

XML creates a separation between content and presentation in an application. Content being the data that needs to be displayed to the client and presentation are the formatting of that data. Using XML to specify the contents in our system would allow us to not tie to any particular presentation implementation. This is useful because we may have to deal to the client heterogeneity of the mobile devices. For example: we may want the templates setting of our server to be accessible from a variety of mobile clients using WML, by storing the contents of these templates in XML, we can specify the transformation that is required to present these contents for separate device by creating a separate XML style sheet for different device types. This way we can guarantee maximum portability in our data.

2.2.2 XML for Communication

As we have discussed in section the XML protocol section in 2.1.2, we can consider using XML as a mean of transferring information between mobile devices. The main advantage of doing this is that we can achieve platform independence and compatibility, since XML is not tied to any type of client. It provides a very simple data representation that can be serialized and transmitted over the network easily.
One draw back from using XML for communication the redundancy that an XML document creates might not be the most efficient way of communication in a particular application. Never the less textual compression algorithms and the binary XML format (WBXML) would provide us with fairly efficient communication.

2.2.3 XML for Configuration

XML could be used also for configuration of applications. We could potentially use XML for configuring our server in part 2 of the project. This provides us with the benefits of platform independence, also decouples our server from it’s configuration tool. Once we have specified a standard interface for the server’s configuration, it is possible to use different configuration tool for the same server.

2.3 Development Tools and APIs

There are various technologies that we can use to develop our mobile communication toolkit. Currently Sun’s Java Micro Edition seems the most promising choice that offers us platform independence and compatibility. We also look into other development platforms that is available for mobile applications.

2.3.1 Sun MicroSystem's J2ME

Sun has last year released it’s Java 2 Micro Edition (J2ME). The package includes the KVM (Kilobyte Virtual Machine) which is a small virtual machine that can run on mobile devices. J2ME supports a subset of Standard Java Platform’s API, it allows developers to write Java application that can be run on mobile devices. J2ME has a ‘profile’, that is a specification of the Java virtual machine functions required to support the J2ME API. J2ME has now KVM that runs on Palm’s OS, Blackberry and Windows CE devices. Motorola and Sony has also announced that they will in cooperate a KVM into their new next generation mobile phones. KVM is minimum configuration that J2ME supports, and developers are allowed to extend this API further to provide a Virtual Machine that works best for their application. J2ME also has the inherited benefit of code portability amongst different devices. Sun has set up an expert group who are coming up with a Java Specification Request (JSR) to define a configuration for these mobile devices, the members includes most large mobile device manufacturers, and it is looking a very promising platform for mobile application development, conforming to Java’s “Write Once run everywhere” slogan.

Figure 5 shows a screen shot of a test application I created to try out the J2ME package. The SDK comes with two sets of API, first the CLDC library which contains all the basic datatypes and classes including stream IO, and some utility classes with data structures such as Hashtables and Vectors. The second set of API is called kJAVA, it contains all the user interface components, some of which you can see in the diagram, including Text boxes buttons and scroll panes.
The process of development is on J2ME is relatively simple. First all the codes are written on a PC using the limited API that is available in the two packages. Then the code is compiled using the ordinary Java 2 Standard Edition compiler. The compiled class files are then preverified which checks that all the classes that you have used does conform to the J2ME standard. Then a Java class specific to the device that you are using, in this case, a class called MakePalmApp is run against the preverified class files to create a Palm application with a .prc format. This can then be installed directly to the palm device through serial link. More information about Sun’s J2ME can be found at http://java.sun.com/j2me/.

2.3.2 Code Warriors

Code Warriors for Palm OS is an Integrated Application Development Environment that consists of a tightly integrated C/C++ compiler, source and assembly-level debugger, assembler and linker. A tool is provided to develop applications on Palm OS using C and C++. This tool is good for developing applications on Palm and Windows CE devices with a very extensive API that other package does not provide. It is also a more mature and established product than J2ME, providing more support for code debugging and a more stable application environment.

The main problem with using this set of tools in this project is that it does not really provide a platforms independence solution. With this being such an important requirement for our mobile communication toolkit, code warrior’s solution is not suitable for our application as we would have to end up implementing a separate version of the communication toolkit for every single mobile platform. Moreover this IDE concentrate on building extensive functionality in the user interface part of the application, it is lacking in pre made solutions for data manipulation classes which the J2ME CLDC provides. Recently Java versions of the IDE is available for mobile applications development, however it is not yet compatible with Sun’s KVM that is rapidly gaining on momentum. More information about this tool can be found at code warrior’s website: .http://www.codewarrior.com/products/palm/.

2.3.3 IBM Visual Age Micro Edition

IBM has taken the idea of J2ME a step further by integrating J2ME support into their popular Java Development Environment Visual Age. Essentially this tool provides a user interface to allow developers to build J2ME applications on a virtual machine that has an extended API of the one provided by KVM. It is particularly convenient at developing user interface through an Drag and Drop tool. This tool mainly extends on the user interface design on mobile application development and would be a useful tool to use in harmony with the toolkit that we are about to develop. More information about this can be found at http://www.embedded.oti.com/.

2.3.4 BlackBerry Java Development Environment

This is an application development environment for the Blackberry wireless device. It has full support with Sun’s J2ME CLDC. It includes an IDE and simulation environment with a set of debugging tools. Also a wireless network simulator for HTTP and TCP connectivity. This API like IBM’s Visual Age Micro Edition would complement J2ME.
2.3.5 XMill

XMill is a new tool for compressing XML data efficiently. It is based on regrouping strategy where XML text strings are grouped together with respect to their meaning. By exploiting similarities between the grouped text strings, then applying a compression technique such as gzip, the compressed XML file can be up to twice smaller than the compressed original file if such grouping strategy has not been applied. If a text based XML protocol is chosen for the way of encoding our mobile communication toolkit, it is possible to incorporate some of the ideas behind XMill for data transfer optimisation.

2.4 Wireless Connectivity

The widespread coverage of the GSM network today has already built into it data connectivity functionality. A whole range of mobile phones supports such data transmission using either a Serial PC Link or an infrared link. There are a lot of new WAP phones that supports viewing of WML page from major phone manufacture including Nokia, Ericsson, Sony, Siemens and Bosch. GSM data connectivity provides up to 9.6Kbps.

At the moment the best wireless connectivity available to the public is the HSCSD (High Speed Circuit Switched Data). HSCSD is a mean of compressing mobile signals by combining simultaneous channels. It is based on the GSM technology that is currently being used over the country. A few new mobile phones in the market today including Ericsson’s R320 and R380, as well as Nokia’s 6210 supports the HSCSD protocol. The phone manufactures claims that the bandwidth can be a much as 48Kbps. Mobile Network Suppliers including Orange and BT announced that it HSCSD services would be available from January 2001, although the stated bandwidth is only 28.8Kbps.

Figure 5: Infrared Enabled Phone with HCSD with data rate up to 28.8 kps

GPRS and 3G aims to provide higher bandwidth data connectivity of up to 2mbps. It features an “Always on”, so there is no need for making a call and the internet can be accessed at the same time as a voice call.

Bluetooth is another up and coming technology growing in popularity. Basically it allows short range transfer of wireless information from mobile devices to “access points”. Access points has provides the data connectivity to device by routing data packets to the appropriate destinations. Connection is achieved using radio frequency, and is aimed at forming local networks of “pico” nets which allows wireless devices within range to...
use the access points for data transmission. At the time of writing this research, Bluetooth has yet few implementations, however it’s key advantage, low cost would guarantee exciting applications to be developed using this technology. Appendix 11.6 compares the characteristics of different wireless networks.

Bearing in mind the different kinds of underlying networks that is involved, it is important to note, it would be ideal if our toolkit does not have to take into account of the underlying network infrastructure that is in place. We can achieve this by concentration on a high level communication protocol that sits on top of the lower network infrastructure.

2.5 Database Server

2.5.1 Microsoft SQL server

MS SQL server is a commercial database server that runs on Microsoft Windows platform. The latest version is MS SQL 2000, it has a lot of interesting functionality, is easy to use and install. The software also comes with a very good database manager called Enterprise Manager, which is a graphical tool for modifying database schema and it’s contents. A text console is also provided called Query Analyser which allows user to enter text SQL queries. The produce however has two main drawbacks, firstly it only works in Microsoft Windows, secondly the package only comes with Microsoft’s ODBC driver for use in applications that was developed using COM. COM which stands for Component Object Model is a Microsoft standard for inter application communications, and is supported mainly by Microsoft Visual Studio package in languages such as C++ and Visual Basic. This means that if I were to develop an application using Java, I would have to purchase a third party JDBC driver. The package is also not freely available.

2.5.2 Oracle SQL server

Oracle database server is a full strength industrial database server that is used in most large-scale commercial applications. The database server is heavy weight and requires a lot of computing resources. It is very difficult to install and administer mainly due to the fact that Oracle has tried to maintain the backward compatibility of newly released database server with previous products. The product runs in the UNIX environment. It does support database connections from nearly all programming languages available. This product however is expensive, and it’s complications make it not really suitable for this project.

2.5.3 PostGrad SQL

PostGrad SQL is a freely available database server that runs on the Linux platform. It supports multiple programming languages and is available at machines in DOC. Different management consoles are also freely available for manipulation of database schema.

2.5.4 MySQL

This is an increasingly popular database server for applications that don’t really need a fully industrial strength database. Mainly for individual users and small businesses. It is freely available in multiple platforms including Linux and Windows. It also provides drivers for uses in nearly all languages as well as “pluggable” management consoles that allows schema manipulation.
2.6 Currently Available Products

The field of wireless information integration is relatively new, and a lot of large corporations are starting to take advantage this potential opportunity for sales. After long investigation, I have found no currently available product that provides platform and application independent communication between mobile devices similar to the toolkit that I aim to build.

I have mentioned in my introduction that synchronisation is currently the most popular way information on mobile devices to be exchanged, I will summaries here some of the products that I have looked into. Also I would summarise the results of my search for products that offers mobile application database querying capability.

2.6.1 Mobile Synchronisation Technology

There are currently various new products that are released in the market that allows data to be exchanged between mobile devices including Lotus’s Mobile Notes, IBM’s Mobile Connect and Palm’s Enterprise Server. I will in this section identify the strengths and weaknesses in each product and compare that with the idea of the system that I am to implement.

2.6.1.1 Lotus Dominal

Lotus Dominal, an integrated messaging and Web application software platform that has recently released a family of products targeted for the Mobile and Wireless sector. They have released Mobile Notes which provides access to Mail, calendar and corporate directory from wireless devices such as WAP phones and PDA using WML.

They also launched EasySync R3.0 for Lotus Notes R5, it provides bi-directional synchronization for Palm Computing devices including the IBM WorkPad PC Companion and PalmPilot Connected Organizers. EasySync enables users to access key information, it can synchronize appointments, meetings, contacts, tasks, memos and emails.

The Lotus Mobile Notes suite performs a similar synchronisation function as the one that I have proposed. However the system relies on current users of Lotus Notes and their proprietary backend system for managing such information.

The system supports network synchronisation, however it does not have the same notion of collaboration and sharing of data amongst multiple users. There is also an Enterprise Edition of Lotus Notes which allows workers in a corporation to share calendar events, but this relies on every user to have the Lotus Notes software, and is geared towards large Cooperation. Mobile Lotus Notes does not support SMS and email notifications. More information about lotus Dominal can be found at http://www.lotus.com.
2.6.1.2 IBM Mobile Connect

IBM’s new Mobile Connect 2.41 enables handheld devices such as IBM WorkPads, Palm, EPOC and Windows CE devices to be integrated into Enterprise Solutions.

Mobile Connect allows organizations to directly transfer information from multiple handheld devices directly to corporate systems, without the need to synchronize via the PC. It enables 2-way relational database synchronization, 2-way file transfer, and the remote installation of applications. IBM Mobile Connect also supports direct synchronization with Lotus Notes and Microsoft Exchange for server based synchronization of e-mail, calendars, contacts and tasks.

Information transfers are supported over any networking infrastructure that supports IP, i.e. Wireless GSM, Internet, as well as standard Local Area Networks (LANs) and Wide Area Networks (WANs). Additionally, the product utilizes a VB Scripting engine that enables a company to "plug" its core business logic components directly into their server. Events handled by these components can then be automatically triggered by the captured data that is being transmitted from the remote hand-held devices. This is similar to the notification functionality that I have identified in section 2.1.2.1 section of my specification.

Mobile Connect uses Certicom’s encryption technology (56 or 128-bit) to provide secure data delivery. Mobile Connect also provides user authentication against an internal configuration or by authenticating users through Lotus Domino or Microsoft Exchange servers.

IBM’s Mobile Connect provides a lot of the important functionality that I have thought of in my specification. However like Lotus’s Mobile Connect, it relies on the user using either Microsoft Exchange Server or Lotus Domino in order for it to work. Both of these software are targeted for large corporations that can install and maintain the expensive servers. Family members and friends who do not have access to these facilities would not be allowed to share their information with the user. Mobile Connect also relies on Microsoft Exchange Server and Lotus Domino server’s ability to share data amongst users. There is no notion of direct connection between devices or a separate component for mobile device to share data between multiple users. Also running the demonstration version of the program shows that the rules that are allowed to specify in the sharing of data is very inflexible. An evaluation version of IBM mobile connect can be found at http://www.ibm.com
2.6.1.3 Palm HotSync Server

The Palm HotSync Server is the standard platform for handheld management, integration and development. The product can integrate with Microsoft Exchange Server conduit (e-mail and calendar), it also has a comprehensive console based management as well as wireless support. HotSync Server is similar to the two software found above, it is also a design that is centric on the Palm’s Architecture. If the EPOC platform wants to take advantage of the functionality, there is not a clearly defined protocol to utilise and the data access level does not have published interface to use. So the system is not very extendible. More information about Palm’s Hotsync server can be found at http://www.palm.com

2.6.2 Database product for mobile devices

Upon research there is no present product that offers mobile device to direct database connectivity. Most present uses are application specific, as developers define their own way to interface a database to mobile devices. They do this by implementing their own input handling code from the device and depending on what the request is from the mobile device, they execute a separate query to the database. Usually the application is a part of a large system, that has already a database server existing clients. Clients can be both web clients as well as other application clients.
3 Specification

Having researched into the different products and technologies related to mobile application development, I would like to draw up a specification for my system. My system consists of two parts:

1. Development of a mobile communication toolkit that allows wireless devices to communicate with each other.
2. Development of a database interfacing server that accepts requests using the communication toolkit and initiate a chain of actions based on the content of the request.

Since the development of the two parts are distinct stages and the toolkit in the first part acts as a component that would be used for communication in the second part, I would consider the specification of each part separately in turns.

3.1 Mobile Communication Toolkit

This communication toolkit allows developers to build applications that runs on mobile devices that can send and receive information with another device. The user of this toolkit has been identified as mobile application developers and their references would be used interchangeably in the remainder of this specification.

3.1.1 System Functions

I start my specification by evaluating the System Functions that I would like to incorporate in my toolkit. System Functions are what a system is supposed to do.

3.1.1.1 Basic Functions

Basic functions are those that are essential for the system, these would be separated into the following two categories. Evident, meaning that function should be performed and the user should be cognizant that it is performed. Hidden, meaning that the function should be performed but it will not be visible to users.

3.1.1.1.1 API to create and manipulate data structure to be sent

Category – Evident

The toolkit shall provide a set of clearly defined interface that allows mobile application developers to create a data structure that contains the contents to be sent. The toolkit also must provide sufficient tools to manipulate this data structure in an easy way. So that when a data structure is received, it is possible to extra and manipulate the contents of the data structure.

3.1.1.1.2 Good authoring tool for content management

Category – Evident

The toolkit shall allow content authors to write their contents easily without any worry of the underlying infrastructure of the toolkit.

3.1.1.1.3 Serialisation of data structure

Category – Hidden
The toolkit shall provide an effective mean of serialising the data structure into the underlying bit stream. This process however should not be directly evident to the developer, although the developer could possible influence the way the serialisation take place through the API.

3.1.1.4 Communication between mobile device
Category - Evident
The toolkit shall provide means to send and receive the serialised data structures in memory into and from an underlying wireless network infrastructure. The toolkit should provides means of a synchronous send and synchronous receive. Ie. The application will block waiting for an input data structure.

3.1.1.5 Can be used in a variety of mobile platforms
Category - Evident
The toolkit shall be able to be used in different mobile platforms. The user should be able to simply install this toolkit into different devices and that the toolkit should behave in a similar way in different devices.

3.1.1.6 Can be used in a variety of network infrastructure
Category – Evident
The toolkit shall be able to operate in a variety of underlying wireless network infrastructure.

3.1.1.7 Easy to debug
Category – Evident
The toolkit shall provide the appropriate tool to debug their own applications when using this toolkit, errors should be traceable by the developer using error handling code.

3.1.1.8 Customisable functionality
Category – Evident
The toolkit shall be able to accommodate extensions to functionality of the toolkit for customised behaviour.

3.1.1.9 Process Tractability
Category – Evident
Developers using the API should be able to trace the exact events that is occurring in the execution of the tool kit.

3.1.1.10 Contents identification
Category – Evident
Developers should be able to identify different content data structure easily, without having to look in depth into the contents. This is very useful as ultimately developers must write some handling code for an received data structure, and if they can recognise the content immediately upon receiving the data structure, we are saving valuable computing resources checking through fields of the contents.
3.1.1.2 Extended Functions

Extended functions are those that are not absolutely essential in order for the toolkit to perform its minimum function. I would aim to complete as many of the extended functions as possible to enhance the functionality of the toolkit. However, it would be possible to begin development of part 2 of this project as soon as the basic functionality is completed, extended functions merely enhances the operations of the toolkit but their interfaces remain the same.

3.1.1.2.1 Compression

Category – Evident

Since speed is the primary concern at the moment for mobile data communication, I would aim to add compression into the communication toolkit to achieve greater efficiency. A concern is the limited processing capability of mobile devices, and it is necessary to take the lowest common denominator for these different device profiles to ensure that we achieve the right balance between data efficiency and processing overheads.

3.1.1.2.2 Asynchronous Receive

Category - Evident

The toolkit shall aim to provide provisions for receiving a data structure asynchronously. The means that a data structure could be received at any time during the running of an application. Again we have to take into account the different hardware profiles that the mobile devices have, an important requirement is multi-tasking capability as this is probably necessary for this functionality.

3.1.1.2.3 End to End Encryption

Category - Hidden

The toolkit shall also aim to provide End to End Encryption for the transmission of the information. Considerations has to be taken on the mobile device’s limited Virtual Machine and processor speed. So a trade off may have to be taken between cryptographic strength and speed.

3.1.1.2.4 Pluggable layers of communication

Category – Evident

Ideally, the toolkit should also provide pluggability of different protocol classes so that we can easily change the protocol that data structures are being sent and received to adapt with technological changes. As seen in our background research, the field of mobile application communication is still very young, and it is not clear yet which protocol format will become the future standard of mobile data delivery. So it would be ideal if new protocols would be plugged into the existing toolkit without changes to applications using it.

3.1.2 System Attributes

I would also like to consider the characteristics that are not functions. These are System Attributes, each system attribute is associated with a set of attribute detail and attribute boundary constraints. Attribute details are the discrete, symbolic values of the attribute. Attribute boundary constraints are mandatory boundary conditions that must be true for the system to be useful.

Here is a table showing the system attributes that are desirable for my toolkit.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Details and Attribute Boundary Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usability</td>
<td>(detail) The API should be easy to use and understand. A clearly defined interface should be presented to the developer.</td>
</tr>
<tr>
<td>Reliability</td>
<td>(detail) The toolkit should be robust, contents send and received should be should not change any content Method procedures should be accurate.</td>
</tr>
</tbody>
</table>
| Security        | (detail) Provide a secure way for sensitive information to be sent across a network.  
(detail) application developer should have control over the degree of security required. |
| Platform        | (detail) Linux, Windows, Palm, Windows CE, Blackberry and most other mobile platform |
| Flexibility     | (detail) Toolkit should provide application developers the maximum degree of control possible. |
| Extendibility   | (detail) Toolkit should be able to operate with different protocol standards. |
| Scaleable       | (detail) The toolkit should minimise the amount of traffic generated to send and receive a piece of information. |

### 3.1.3 Relationship between System Function and System Attributes

Now it is possible to relate the system functions that I have drawn up against the system attributes that are desirable in my toolkit. Figure 6 on the next page shows a list of system functions and system attributes. I have drawn a line joining a system function with a system attribute if by satisfying the system function on the right, we achieve the system attribute stated on the left. The doted line signifies the overlapping of a system function across several system attributes.

From figure 6, we can see that all system attributes spans across the system function space that I have listed, this means that my functionality proposed has fulfilled all the attributes that I can think of that is desirable in the toolkit. It is also interesting to note that 5 of the system functions that I have identified related to fulfilling the usability attribute of the system, this is desirable because usability is one of the most important aspect of designing a toolkit.

Another interesting point to note from the diagram is that by implementing the basic functionality that I have identified in section 3.1.1.1, we basically cover all system attribute except for the flexibility attribute. This means that we have more or less scoped the basic functionality right, and by implementing the extended functionality we would be providing mainly more flexibility, extendibility and scalability to the toolkit.
3.2 Database Interfacing Server

The database interfacing server’s main purpose is to give mobile applications the capability to query database at high level, this means that mobile application does not really have to follow a specific standard or format to submit a data structure representing a query. In fact the mobile application does even have to consciously know that it is submitting a database query. All that the mobile application should have to do is to send a data structure with the appropriate parameters for the information that is required. Definitions of templates at the server is then responsible to map these high level data structures to low level database operations. This way we can completely decouple the mobile application with the details of conventions used in the server.

3.2.1 System Function

To understand what I am trying to build here, I have listed the basic and extended system functions that I aim to implement in this server.
3.2.1.1 **Basic Function**

The following are the functionality that I consider essential for this server. Similar to the toolkit, I have categorised these functionality into evident or hidden functions allowing me later to plan my implementation and perhaps group similar functionality together in development.

3.2.1.1.1 **Ability to send and receive data structure from mobile device**

Category - Evident

The server shall have the ability to send and receive data structures containing request and query replies to and from mobile applications. This should be automatically achieved upon completion of the basic functionality of the communication toolkit in part 1.

3.2.1.1.2 **Connection with a database server**

Category – Evident

The server shall be able to connect to a database server, and that the settings of the database connection should be configurable using a user interface.

3.2.1.1.3 **Definition of data structure that server is allowed to accept**

Category – Evident

Application developers shall be able to define the type of data structures this server is allowed to accept using an easy to use user interface.

3.2.1.1.4 **Manipulation of database schema**

Category – Evident

Application developer should be able to extend the database schema easily to accommodate new application data and functionality.

3.2.1.1.5 **Template Definition using a “drag and drop” interface**

Category – Evident

Application developer should be able to define and manage template definitions using a drag and drop interface. Template definitions should provide the ability to map the data structures defined in 3.2.1.1.3 to data base operations.

3.2.1.1.6 **Long term storage of server configurations**

Category – Evident

The information defined in sections 3.2.1.3 to 3.2.1.5 should all be stored into a long term storage location. Possibly in the database or in a configuration file.

3.2.1.1.7 **Handling of data structure request**

Category – Evident

The server should be able to receive data structures that was defined in section 3.2.1.1.3 and handles the requests based on the template definition in section 3.2.1.1.3. It must also return any result required back to the mobile device, if this is defined to be so in the templates.
3.2.1.2 Extended Function

The following functionality are desirable in server but is not essential.

3.2.1.2.1 Decoupling of server and configuration tool

Category – Evident

The roll of the server is to handle requests based on the templates, in a production environment it would probably be used by the system administrator. Whilst the configuration tool is for the definitions of the templates, and would be used by application developers. So it would be nice if our server and it’s configuration tool is decoupled, this gives us greater flexibility also in defining the type of configuration clients we can use.

3.2.1.2.2 Allow encoding of conditionals and loops in the template

Category – Evident

If we can allow the template to define conditionals and loops, then it would be much easier for handling list data structures and also gives the server greater power to perform more complicated operations.

3.2.1.2.3 Dynamic Configuration

Category – Evident

The templates should be alterable whilst the server is running. And the alterations should be effective immediately. This means that we can change the templates (business logic) of the server without having to bring the server down from use.

3.2.1.2.4 Traffic and access statistics

Category – Evident

The server shall provide the appropriate logging functionality of access information over a period of time, the log file shall be stored in a long term storage location.

3.2.2 System Attribute

The following table shows the system attributes and their boundary conditions that I would like my system to have.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Attribute Details and Attribute Boundary Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usability</td>
<td>(detail) The server should be configurable using an easy to user interface.</td>
</tr>
<tr>
<td></td>
<td>(detail) The server should present to the user wizards, flow diagrams to assist the understanding of template definitions</td>
</tr>
<tr>
<td></td>
<td>(detail) The server should allow user to chronologically define the sequence of template configurations, similar to the order in which the server handles a request.</td>
</tr>
<tr>
<td>Security</td>
<td>(detail) The server should reject any data structures that is not defined in the template. So that malicious damages are not caused to areas of the database that have not been configured to be accessed.</td>
</tr>
</tbody>
</table>
### 3.2.3 Relating System Function and System attribute

The following diagram shows the relationship between the server’s system function and system attribute using the same conventions as those defined in section 3.1.3.

![Diagram showing the relationship between system attributes and functions of server](image)

**Figure 7: Relationship between system attributes and functions of server**
It can been seen that our system function covers nearly the whole system attribute space that we have defined. The only system attribute not covered by a system function is platform, so this has to be added to one of the requirement in our specification. The figure shows that we have scooped our basic system function right since, 7 basic functions identified covers 9 out of 10 of the system attributes that there are. The three extended system functions that we have identified adds to the maintainability and extendibility of our server.

3.3 Use Cases

Use cases are textual narrative descriptions of the sequence of events of an actor (an external agent) using a system to complete a process. Since my server involves more complicated interactions with a set of different users, I have decided to use them to help me to further improve on my understanding of the processes that will be involved. I would adopt an actor-based approach in identifying use cases of the server.

3.3.1 Identifying Actors

First I identify the actors of my system, these are entities external to the system who in some way participates in the story of the use case. The actors that I have identified includes:

3.3.1.1 Mobile Application Developers

These include individuals who develop applications that runs on mobile devices and would like to use our system for building communications and database capabilities in their system.

3.3.1.2 System Administrator

These include individuals who are responsible for installing, configuring and monitoring our server to ensure that it is functioning as required. He or she may also need to access statistics and information about server’s activity over a period of time.

3.3.1.3 Mobile Application Users

These are individuals who uses the mobile applications and issues the requests to the server.

3.3.1.4 External systems

These are systems other then mobile applications who want to request for information from the server’s database.

3.3.2 Use Cases for the System

Having identified the actors of our system, it is possible to find out the uses cases of this system by identifying the processes that each actor initiate or participate in. First I would explain the terms that I would use in classification of my use cases. Then I would present with a scenario which covers the main uses of this system.

3.3.2.1 Definitions of terms
The following use cases are presented in a "conversational" style between the actors and the system. I have categorised the use cases as either primary, secondary or optional. Their descriptions are given below.

1. **Primary use cases**: represents major common processes.
   **Secondary use cases**: represents minor or rare processes.
   **Optional user cases**: represents processes that may not be tackled.

In the type field of each use case, there is a second field that represents whether the use case is an essential use case or a real use case. Their definition are given below.

2. **Essential use cases**: are use cases that are expressed in an ideal form that remains relatively free of technology and implementation details. Where the use case does not give much information about the design decisions, instead an abstract description of events are given.
   **Real use cases**: are those that concretely describes the process in terms of it’s real current design. It is usually committed to specific design, input and output technology.

The type field would help me to understand more about the nature of the use case, this allows me to make design decisions when planning the implementation of the functionality referenced by this use case.

I have also decided to include the cross references of the system functions that has to be developed in order for the use case to work. This will help me to analyse and prioritise the functions that has to be developed in order to benefit the most from the user and processes point of view.

### 3.3.2.2 A Scenario
Since the main purpose of my system is to enable communication and database capability in mobile application, and that the system functions that I have identified have to be carried out in a strict chronological order, I would I will illustrate my use cases in a typical mobile application development scenario consisting of distinct steps of processes making up my use cases. Let’s consider the following scenario:
Scenario title: Developing a new template for a searching stock prices

Scenario Description: An application developer has created a new application on a wireless device for searching of stock prices given the stock symbol or number.

He has access to a database which stores up to date information of individual stock quotes. The stock quotes comes from a third party information feed.

He believes that if he can provide up to date stock quote data in his mobile application, it would become hugely popular to the large amount of people who has interests to up to date transaction information in the stock market around the globe.

He does not want to spend too much time building his own communication procedures and intermediary node that executes the database query for the mobile devices.

He decided to use our toolkit and server to incorporate this searching capability to his mobile application.

This scenario would generate the following use cases in chronological order.

- **Developer Start up of server configuration tool and connect to a database. Use case 3.3.2.3.**
- **Developer add a new type of requests that the server can accept. Use Case 3.3.2.4.**
- **Developer extends the database schema to include any new information required to be accommodated. Use Case 3.3.2.5.**
- **Developer constructs a template mapping for a newly created data structure. Use case 3.3.2.6.**
- **Mobile User uses the application created by developer to search stock quotes Use Case 3.3.2.7.**
- **Administrator wants to access the server’s access log for analysis of traffic information. Use case 3.3.2.8.**

### 3.3.2.3 Developer Start up of server configuration tool and connect to a database

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Developer Start up of server configuration tool and connect to a database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>Mobile Application Developer</td>
</tr>
<tr>
<td>Purpose</td>
<td>Allow mobile application developer to change the configuration of a server.</td>
</tr>
<tr>
<td>Overview</td>
<td>Developer starts up application, defines the database connection</td>
</tr>
</tbody>
</table>
Below is a course of events that occur in chronological order during the period of this use case.

### Course of Events

<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. This use case begins when user starts up configuration tool</td>
<td>2. System presents to the user the main interface of the configuration tool. All configuration activity should be disabled to start with, presenting the user with the database settings screen.</td>
</tr>
<tr>
<td>3. User enters the appropriate setting for database to connect, including user name, password, port and location of database. User presses connect button.</td>
<td>4. System attempts to establish connection with the database using the settings entered and reports back to the user of its progress.</td>
</tr>
</tbody>
</table>

### 3.3.2.4 Developer add a new type of requests that the server can accept

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Developer add a new type of requests that the server can accept.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>Mobile Application Developer</td>
</tr>
<tr>
<td>Purpose</td>
<td>Developer want this server to recognise the new stock quote request data structure.</td>
</tr>
<tr>
<td>Overview</td>
<td>User opens wizard for defining a new request type. He then uses the designer to define how the encapsulated data is laid out and the data type to expect from each field. Upon completion user press save button to save the configuration to a database.</td>
</tr>
<tr>
<td>Type</td>
<td>Primary</td>
</tr>
<tr>
<td>Cross References</td>
<td>Functions: 3.2.1.1.2, 3.2.1.1.3, 3.2.1.2.1, 3.1.1.1.1, 3.1.1.1.3</td>
</tr>
</tbody>
</table>

### Course of Events

<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Use case begins when user opens up screen for data structure definition in a configuration tool after database connection has been established.</td>
<td>2. System presents to the user a list currently defined data structure, and menus to insert, edit or delete data structure.</td>
</tr>
<tr>
<td>3. User clicks onto the insert button</td>
<td>4. System starts up a wizard for data structure definitions.</td>
</tr>
<tr>
<td>5. User first enters a name and description to identify the data structure in subsequent edits. User press next button.</td>
<td>6. System presents user with a user interface to define the data structure and...</td>
</tr>
</tbody>
</table>
7. User uses interface to define new data structure. And presses the save button.
8. System saves new data structure definition to the database, then gives confirmation to the user of progress.

### 3.3.2.5 Developer extends the database schema to include new stock information required to be accommodated

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Developer extends the database schema to include any new information required to be accommodated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>Mobile Application Developer.</td>
</tr>
<tr>
<td>Purpose</td>
<td>Extend and modify the database to accommodate new functionality required by mobile application.</td>
</tr>
<tr>
<td>Overview</td>
<td>User opens up dialog to edit schema, he then types in the SQL command to create a table in the database called stock_quotes to accommodate the new information.</td>
</tr>
<tr>
<td>Type</td>
<td>Primary, Essential</td>
</tr>
<tr>
<td>Cross References</td>
<td>Functions: 3.2.1.1.4, 3.2.1.1.2, 3.2.1.1.6</td>
</tr>
</tbody>
</table>

#### Course of events

**Actor Action**

1. Use case begins when user starts up server configuration tool connected to a database, then clicks onto the dialog for editing database schema.
2. System presents user with a space to enter string command, a list of recently submitted SQL queries and a message box to report feedback to user.
3. User enters SQL statement to create a table called stock_quote, then press the execute button.
4. System checks the syntax of the statement, no error is found, system executes statement and returns result in the message box.

### 3.3.2.6 Developer constructs a template mapping for a newly created data structure

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Developer constructs a template mapping for a newly created data structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>Mobile Application Developer</td>
</tr>
<tr>
<td>Purpose</td>
<td>Allows user to define the sequence of actions that the server would take upon receiving a particular data structure.</td>
</tr>
<tr>
<td>Overview</td>
<td>User opens up wizard to construct a new template, enters a name for the template, using the drag and drop interface user selects constructs definitions for template. User save template to database</td>
</tr>
<tr>
<td>Type</td>
<td>Primary, Element</td>
</tr>
</tbody>
</table>
### Course of events

<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Use case starts when user starts up stock quote application on mobile device</td>
<td>2. System presents to the user a dialog for entering the name of this template configuration.</td>
</tr>
<tr>
<td>3. User enters name and press next.</td>
<td>4. System presents a drag and drop interface, that allows the construction of a map of action given the an input data structure.</td>
</tr>
<tr>
<td>5. User constructs a flow diagram to represent how the template. Upon completion user press save.</td>
<td>6. System saves the whole map to the database.</td>
</tr>
</tbody>
</table>

#### 3.3.2.7 Mobile User uses the application created by developer to search stock quotes

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Mobile User uses the application created by developer to search stock quotes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>Mobile Device User</td>
</tr>
<tr>
<td>Purpose</td>
<td>Allows use to search stock quotes information using wireless connection. Stock information is stored in a database.</td>
</tr>
<tr>
<td>Overview</td>
<td>User enters criteria for searching stock quotes, submits request to server. Upon receiving request, server looks up template definitions in memory, sever query database based on criteria received, and sends result back to mobile device.</td>
</tr>
<tr>
<td>Type</td>
<td>Primary, Essential</td>
</tr>
<tr>
<td>Cross References</td>
<td><em>Functions:</em> All of 3.1.1.1, 3.2.1.1.1, 3.2.1.1.2, 3.2.1.1.3, 3.2.1.1.5, 3.2.1.1.6, 3.2.1.1.7</td>
</tr>
</tbody>
</table>

#### Course of events

<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Use case starts when user starts up stock quote application on mobile device</td>
<td>3. Mobile application constructs a data structure representing the search and it’s criteria, it then sends this document to the server at the predefined address by using a wireless network.</td>
</tr>
<tr>
<td>2. User enters the screen for searching stock quotes. User enters criteria to search and press search button.</td>
<td></td>
</tr>
</tbody>
</table>
4. Server receives data structure, look up in it’s configuration on whether to accept the data structure. Finds in the configuration that a template is available.  
5. Server applies the request to the stored template and performs a database operation to find if there are entries of stock that matches the criteria.  
6. Using results returned from database query, server constructs a reply and sends it back to the mobile device.  
7. Mobile device receives result and display it on the screen to user.

3.3.2.8 Administrator wants to access the server’s access log for analysis of traffic information

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Administrator wants to access the server’s access log for analysis of traffic information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actors</td>
<td>System Administrator</td>
</tr>
<tr>
<td>Purpose</td>
<td>Allows system administrator to analyse access information of server.</td>
</tr>
<tr>
<td>Overview</td>
<td>Administrator selects logging method and logging options, and starts the logging process. After a period of time, administrator opens up log file for analysis.</td>
</tr>
<tr>
<td>Type</td>
<td>Primary, Essential</td>
</tr>
<tr>
<td>Cross References</td>
<td>Functions: 3.2.1.1.6, 3.2.1.2.3, 3.2.1.2.4</td>
</tr>
</tbody>
</table>

Course of events

<table>
<thead>
<tr>
<th>Actor Action</th>
<th>System Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Use case starts when user starts up server and selects logging from menu</td>
<td>2. System loads setting for logging and presents to user current settings</td>
</tr>
<tr>
<td>3. User enters the desired options for logging and starts logging process.</td>
<td>4. System starts start logging any future access to a file using the desired option.</td>
</tr>
</tbody>
</table>
3.3.3 Use Case Coverage

Now that all the system functions identified in sections 3.1.1 and 3.2.1 have all been allocated to use cases, this is an indication that my six use cases are a good representation of the processes that are involved in my system. The cross reference section in each use case would be traceable later on in the implementation and testing phrase.

3.3.4 Use Case Diagram

To further understand the relation between actors and use cases in my system, I have constructed the use case diagram for my system. The system boundary includes the mobile communication toolkit, the interfacing server and server configuration tool.

![Use Case Diagram of my system](image)

The diagram above shows the use case diagram of my system, and the surrounding figures represent actors of my system. The diagram shows that most of my use cases consists of the actor being the application developer.

An important point that becomes apparent after construction of the diagrams above is that the use case of submitting a request to the server can be acted upon by either mobile devices or external systems. This means that they both shares a subset of the same functionality provided by the server system. This also indicates that it is possible to build a common interface for the requests from both actor, this is particularly helpful in satisfying the extendible and platform independence system attributes that I have identified earlier on.

3.3.5 Scheduling Use Case

Although it is possible to rank and prioritise the above use cases by judging on different qualities that each use case would bring to our system, such as impact on design, insight added, amount of research required to help us schedule our development stages, I’ve found that this is not appropriate in this case. This is because a particular use case above would not mean much if it’s predecessor use case has not been performed. So naturally the above us cases already fall into a chronological order of development.
3.4 System Models

I have drawn here, the contextual model and the data flow model to provide a graphical representation of my system from two different perspectives.

3.4.1 Context Models

The following context models shows the boundaries of my system and how it will interact with it’s environment.

![Context Model Diagram]

The diagram shows that system administrator and the database would be interacting directly with the server. Whilst mobile application developers interact with the server via the configuration tool. Mobile Devices and external system interact with the server via the mobile communication toolkit.

3.4.2 Data Flow Model

Data flow model shows how data is processed by a system. The following is the data flow model for my system during the scenario described in section 3.3.2.2.

The notation used is as follows: ellipses represents processing steps, arrows represent flows of data, name of data that is flowed is described by the annotation on the arrows and rectangles represent data stores or data sources.
The diagram gives us two pieces of useful information that would help us later when designing our system. Firstly we can clearly identify the input and output that is required at different processing steps, this would give us an indication when implementing the functionality kind of input does the function takes and what would be the output of the function. Secondly the diagram tells us what information we have to carry along between different processes, this would be useful when specifying the architecture of out system later on.

Figure 10: Data flow diagram of scenario
Part 1 - Design of Mobile Communication Toolkit

This section would describe the design of part 1 of this project, development of the platform independent mobile communication toolkit.

4.1 Design Decisions

This section would describe the reason why I have decided to choose specific technology to implement this toolkit in preference to the alternatives available which was discussed in the background section of this document.

4.1.1 Choice of Environment

We have looked into two possible development environments that we could build our toolkit from in section 2.3, the Java 2 Micro Edition and Code Warrior C++. Despite the greater reliability and capability of the Code Warrior C++ development environment, J2ME’s main advantage of platform impendence provides us with the perfect solution for our toolkit as we will not have to implement separate versions of our toolkit in different platform. This advantage is even more important in today’s world of mobile computing, where there is a large diversity of platform architecture available. I have also tested out the KVM (kilobyte virtual machine) on a Palm device, and have found that the existing CLDC package should provide us with sufficient utilities to implement our set of toolkit efficiently.

4.1.2 Protocols Comparison

The follow matrix summaries and compares the features that each XML protocol we have investigated in section 2.1.2 has.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Status</th>
<th>XML Schema</th>
<th>DID</th>
<th>Flexed/Automatic</th>
<th>Explicit/Implicit</th>
<th>Extensibility</th>
<th>Compression</th>
<th>RPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML-RPC</td>
<td>Deployed</td>
<td>No</td>
<td>Yes</td>
<td>Fixed</td>
<td>Explicit</td>
<td>No</td>
<td>Text</td>
<td>Yes</td>
</tr>
<tr>
<td>SOAP</td>
<td>Coded</td>
<td>Yes</td>
<td>No</td>
<td>Automatic</td>
<td>Brth</td>
<td>Yes</td>
<td>Test</td>
<td>Yes</td>
</tr>
<tr>
<td>WML</td>
<td>Deployed</td>
<td>No</td>
<td>Yes</td>
<td>Fixed</td>
<td>Explicit</td>
<td>No</td>
<td>Test</td>
<td>No</td>
</tr>
<tr>
<td>SYNCML</td>
<td>White Paper</td>
<td>Unknown</td>
<td>Unknown</td>
<td>Fixed</td>
<td>Explicit</td>
<td>No</td>
<td>Test</td>
<td>No</td>
</tr>
<tr>
<td>WBXML</td>
<td>Spec Approved</td>
<td>Yes</td>
<td>Yes</td>
<td>Automatic</td>
<td>Brth</td>
<td>Yes</td>
<td>Binary</td>
<td>Yes</td>
</tr>
</tbody>
</table>

An explanation of each of the features described in the column heading is given in appendix 1. Please refer it for more detailed explanation of each feature. From the table we can see that SOAP would be a good choice as a format to encode our XML documents for the mobile communication toolkit. It provides us the application independence and extendibility that we require, as well as supporting remote procedure call that would be useful for wireless applications. SYNCML although sounded very promising from the released white paper, and with the application on mobile devices in mind in design, it
should provide us a good choice for future compatibility with other mobile devices. However the
standard is still in its infancy, and no approved specification of the standard being available means that
it is not a possible choice for our encoding standard. WBXML is the only protocol that encodes data in a
binary form, and has the extensibility that SOAP provides, it is also an approved standard by WAP
forum, I think that it would be a safe choice to use WBXML as our communication standard but in our
implementation we should bear in mind the possibility of a different emerging standard and have the
framework decoupled with the actual standard that we are using for transmitting XML documents.

4.1.3 Choice of parsing framework
In section 2.1.3 we looked at two parsing frameworks, SAX and DOM. Although SAX provide a
more efficient framework of parsing by minimising the amount of memory that is required to store the
document in memory, I think that I would choose to use DOM as the method of parsing. This is because
DOM provides us with the greater flexibility, and the ability to manipulate the whole parsed document
together in a hierarchical fashion. Also application developers may find it much easier to create their
applications when different parts of the documents could be reference at any time after the document is
parsed, rather than individual document entities becoming available as the document is parsed as is the
case in the SAX framework.

4.2 WAP Binary XML Encoding Basics
The complete structure of an WBXML document would be found in BNF form in appendix in section
11.2. In this section I would give a brief overview of WBXML encoding. More details about WAP binary
XML content format encoding could be found at the WWW consortium website at http://w3.prg/TR/wbxml.

4.2.1 Introduction
WAP which stands for wireless application network is a result of continuous work to define an industry-
wide specification for developing applications that operate over wireless communication networks.
WBXML is a standard that defines a compact binary representation of the Extensible Markup Lineage
(XML). The binary XML format is designed to reduce the transmission size of XML documents to allow
efficient use of XML data on narrowband communication channels.

The binary format caters for the encoding of the element structure of XML with out losing any
functionality or semantic information. Meta information including DTD (document type information) and
conditional selections is removed upon conversion of document to binary format.

4.2.2 Encoding Integers and Characters
Integers are encoded as a series of octets, where the most significant bit is the continuation flag,
indicating that an octet is not the end of the multi-byte sequence. Value of one indicates that the this octet
is not the last byte of the integer. The remaining seven bits in each octet are encoded as scalar values.
Network byte and bit ordering is big-endian.

The encoding of all strings in the XML binary content format is specified by the container level meta
information. A char set declaration included in the header information of the document specifies the
format of the character encoding.
4.2.3 XML Header Information
A binary XML document consists of two parts, a header and a body. The encoding of header information in binary XML is relatively straightforward and is summarised with the following table.

<table>
<thead>
<tr>
<th>Header Content</th>
<th>Purpose</th>
<th>Encoding Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version Number</td>
<td>Specifies the WBXML version</td>
<td>Encoded as an 8 bit unsigned integer. First four bit encodes the major version minus one, last four bit encodes the minor version.</td>
</tr>
<tr>
<td>Document Public Identifier</td>
<td>Identify well know document type contained within the WBXML entity</td>
<td>Encoded as multi byte positive integer representing the identifier. If predefined number is not available, can be encoded as string.</td>
</tr>
<tr>
<td>Charset</td>
<td>Represents the character set in this XML document, uses the IANA assigned MIB number for a character set.</td>
<td>MIB number as multi byte integer.</td>
</tr>
<tr>
<td>String table</td>
<td>Used to encode the strings that would be needed in the WBXML document. Later references to entries to this string table is encoded as offset from the start of the table. (more details later)</td>
<td>Multi byte integer representation of the length of the string table, followed by a list of bytes representing the string entries.</td>
</tr>
</tbody>
</table>

4.2.4 Token Structure
The body of a binary XML document consists of a set of tokens. These are octets of codes that have different meanings depending on the context in which it is used. Tokens are classified as global tokens or application tokens.

4.2.4.1 Application Token
Application tokens are split into two overlapping code spaces, tag code space and attribute code space. A given application token will have different meaning depending on whether it represents a token in the tag or attribute code space. The tag code space represents a specific tag name in the document, whilst the attribute code space represents an attribute of an element in the document. The numeric range of the attribute code represents whether the code represents attribute prefixes or attribute values. Details of encoding in the tag and attribute space could be found in appendix 11.3.

4.2.4.2 Global Token
Global tokens have the same meaning and structure in all code spaces and in all code pages. It is used to encode inline information such as Strings (from string table references), DTD extension tokens, inline opaque data, character entities, XML processing instructions, literal and control codes. Details of each encoding could be found in appendix at 12.4.
4.2.4.3 Code Pages
Each code space is further split into a series of 256 code pages to allow for future expansion of the
well-know codes. A single token (SWITCH_PAGE) switches between the code pages.

4.2.4.4 Encoding Example
Appendix 11.5 shows an example of encoding XML document into it’s binary counterparts using
the WAP standard. Note that the example uses both predefined tag and attribute tokens as well as the
use of string tables to encode unknown tags and attributes.

4.3 The Visitor Pattern
Central to the design of our toolkit, lies the visitor pattern. The visitor pattern is a behavioural design pattern,
it represents an operation that can be performed on the elements of an object structure. Visitor pattern allows us
to define new operation for each element in the structure without changing the classes of the elements on which it
operates. As you will see later the visitor pattern is used in the design of the encoding procedures, parsing
procedures and for pretty print of our document hierarchy when implementing our DOM framework.

The following diagram shows how the visitor pattern is applied in my toolkit, UML notation is used.

![Diagram showing application of visitor pattern in toolkit](image-url)
The visitors in my applications includes the WBXML encoder, WBXML parser as well as the DocumentPrinter for printing of the document in a console. The main advantage that the visitor pattern gives to my application is that it can help me to achieve a good separation between XML construction and manipulation part of my toolkit with the underlying protocol that is being used for sending and receiving the XML document. If for example we wish to support a new protocol for transmission of our XML documents, all we have to do is to implement a new concrete subclass of the encoder and parser class. We would not need to change any of the element in the element and attribute class because the reference of Visitor passed into the accept method determines which visitor to call.

Another consequence of using the visitor pattern is that it provides a common interface to encode or parse both an element and attribute, this means that we can build a class which concentrates on the tree traversal behaviour without having to worry about whether an element or attribute is being encoded or parsed.

As well as the advantages, it should also be mentioned that the use of the visitor pattern makes adding any concrete subclass of entity more difficult. This is because it will be necessary to extend every concrete visitor to include a method to encode and parse the new structure. But since it is unlikely that we will be extending the semantics of XML, this disadvantage should not be apparent in the implementation of our toolkit.

4.4 Binary XML Encoder Design Considerations

This section would describe my design for the binary XML Encoder in my toolkit. I will look at the different algorithms that are available for traversing through the document tree to encode the entities in the document into an output stream for wireless network transmission.

4.4.1 Order of tree traversal

I will illustrate the different tree traversal strategy there are for when encoding and parsing XML tree.

4.4.1.1 Pre-order tree traversal

This is when the tree is traversed in the following fashion:

1. visit the root
2. traverse the left subtree in preorder
3. traverse the right subtree in preorder

This would contribute to the following tree traversal order in an XML tree. The numbers on each node specified the order of traversal.
This strategy provides a simple way to traversing through the tree and is especially good for trees that has a large depth.

### 4.4.1.2 In-order tree traversal

In in-order tree traversal, the tree is visited in the following order.

1. traverse the left subtree in inorder
2. visit the root
3. traverse the right subtree in inorder

This strategy gives rise to the following tree traversal order in the example sub tree.

![In-order tree traversal diagram](image)

**Figure 13: In-order tree traversal**

### 4.4.1.3 Post-order tree traversal

1. traverse the left subtree in postorder
2. traverse the right subtree in postorder
3. visit the root

This strategy gives rise to the following tree traversal order in the example sub tree.

![Post-order tree traversal diagram](image)
4.4.1.4 Breath First Traversal

A breadth-first traversal of a tree starts at the root of the tree. It next visits the children, then the grand-children and so on. The following show an example of the traversal.

One of the main reason of the wide spread use of this strategy is because that it is easy to implement, and requires the least amount of buffer space to store temporary results from multiple level of nesting.

4.4.2 Recursive Tree Traversal

The most common way of tree traversal is to use recursion. The following code shows the standard way to achieve this.

```c
void recursive_traverse(node *pSubTree) {    //pre-order recursive traverse through a tree
    if(pSubTree){    //if node exist (not null)
        visit(pSubTree),    //visit current node
        recursive_traverse(pSubTree->leftTree); //visit left branch
        recursive_traverse(pSubTree->RightTree); //visit right branch
    }
}
```
This algorithm is simple and concise, it provides a fast and effective way to implement different tree traversal process. It takes advantage of the natural hierarchical arrangement of the tree nodes and uses back-tracking and recursion to do the rest of the work of visiting all the other nodes. However this method has one major draw back, it is that it may take a large amount of memory to run if a particular branch is deep, this is because the recursive procedure recurse_traverse does not returns until all the sub nodes has completed their execution, and meanwhile all the intermediate results are stored on the stack, this may cause a stack overflow.

4.4.3 Non Recursive Tree Traversal

Instead of using recursion, it is possible to traverse through the tree using iteration. The following is the algorithm.

```c
Void traverse(node *pSubNode){
    stack.push(pSubTree);    //first push node in stack
    while(!stack.empty()){
        pSubTree = stack.pop();  //pop node from top of stack
        visit(pSubTree);       //visit this node
        if(pSubTree-> rightTree) //if this node has a right sub node,
            stack.push(pSubTree->rightTree); //push right node to stack
        if(pSubTree-> leftTree)  //if this node has a left sub node
            stack.push(pSubTree->leftTree); //push left node to stack
    }
}
```

The algorithm uses a large while loop to traverse from one node in the tree to the next, during each iteration a node is taken from the top of the stack, then the node is visited before adding it’s children to the top of the stack. The stack acts as a holding list for the children of visited nodes that has yet to be visited, the list is also arranged in order of visit.

This algorithm has the advantage over it’s recursive version when the tree structure is deeply nested, however it is not as elegant to implement, specially when a tree node has many children.

4.5 Encoder Design

This section describe the details of the design of my WAP binary XML encoder.

4.5.1 Choice of Traversal Strategy

The encoder uses a breath-first, recursive tree traversal strategy. This strategy was chosen because it offers a simple and efficient way of encoding the XML tree, also at the same time with the minimum amount of temporary buffer being needed, since each tree level is being encoded in order. The recursive
version of the strategy is chosen in preference to the iterative strategy because nearly all applications of this toolkit in mobile applications would not need a highly nested document structure, instead most documents should have a large breath, containing different entities of small data to communicate.

### 4.5.2 Class Diagram

The following diagram shows a class diagram of the design of the encoder.

![Class Diagram of Encoder](image)

The concrete implementation of the WAP binary XML encoder would extend an abstract class called `encoder` which has the interface of for encoding an element and an attribute. The encoder utilise the visitor pattern, and is responsible for recursively traversing through the XML tree in a breath first fashion. For every entity that the encoder visits, the encoder would call the encode method of thee entity so that the appropriate representation of the entity is serialised in the WBXML standard and written into a buffer which would subsequently be written to the underlying network streams.

The above design allows us to treat both element and attribute as an entity in the XML tree, this means that the tree traversal algorithm does not have to be aware of which it is encoding, it can therefore
concentrate on the tree traversal strategy, and call the common encode method of entity it passes through as the tree is traversed.

This design also allows us to abstract away the underlying implementation of the encoding method from the tree entity structure. It is therefore easier to “plug” a need encoding standard to the framework without changing the underlying structure of the toolkit. This allows us to satisfy system function 3.1.1.2.4 “Pluggable layers of communication layer” requirement that we have specified.

4.6 XML Framework Design

I have discussed in section 4.1.3 that the DOM framework for XML manipulation would be used in my toolkit. The following describe how I designed my toolkit, according to the DOM Level 1 specification.

4.6.1 Document Object Model (Core) Level 1

Document Object Model is an API that defines the logical structure of documents and the way an XML document is accessed and manipulated. A specification of the interface could be found at http://www.w3.org/TR/REC-DOM-Level-1/level-one-core.html.

In DOM Level 1, the fundamental interfaces includes the following document entities.

<table>
<thead>
<tr>
<th>Entities</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOMException</td>
<td>Exceptions that could be raised during parsing of an XML document.</td>
</tr>
<tr>
<td>Document Fragment</td>
<td>Light Weight Document Object for document manipulation</td>
</tr>
<tr>
<td>Document</td>
<td>XML Document, provides primary access to document data</td>
</tr>
<tr>
<td>Node</td>
<td>The Node interface is the primary datatype for the entire Document Object Model. It represents a single node in the document tree</td>
</tr>
<tr>
<td>NodeList</td>
<td>The NodeList interface provides the abstraction of an ordered collection of nodes, without defining or constraining how this collection is implemented.</td>
</tr>
<tr>
<td>NamedNodeMap</td>
<td>Used to represent collections of nodes that can be accessed by name.</td>
</tr>
<tr>
<td>CharacterData</td>
<td>The CharacterData interface extends Node with a set of attributes and methods for accessing character data in the DOM</td>
</tr>
<tr>
<td>Attr</td>
<td>The Attr interface represents an attribute in a Element object</td>
</tr>
<tr>
<td>Element</td>
<td>Representation of a particular entity in the XML document.</td>
</tr>
<tr>
<td>Text</td>
<td>The Text interface represents the textual content (termed character data in XML) of an Element or Attr</td>
</tr>
</tbody>
</table>

4.6.2 Platform Considerations

DOM Level 1 provides the complete API for manipulation of an XML document, however it is necessary to take into consideration the limited processing capability of the mobile devices that may use
the toolkit. Also we need to take into account of the slimmer Java API that is available on most mobile devices. Another consideration that I have to take into account is the fact that the WAP binary XML specification does not cater for the encoding of meta level information, therefore information such as document comments document type definitions could not be easily encoded into the binary stream without breaking the standard of WBXML that we are trying to conform to. This means that some kind of tradeoff must be made between protocol conformance and conformance to the DOM specification. I have decided that the DOM interfaces must have to be compromised to take into consideration of the limited processing capability, limited Java API and limited protocol features that I have. However I would try to stick with the framework that DOM has provided. I would try to follow its parsing mechanism, the element entities and their hierarchical arrangements and the way these elements are manipulated and accessed.

4.6.3 XML Document and Element manipulation

At the core of the DOM concept, is the idea of elements representing individual entities in the XML document, and their hierarchical arrangement into parents and children that allows developers to access and manipulate the document in a simple fashion. The following example XML document demonstrates a DOM representation of a list of stock quotes information in an XML document.

![DOM Representation of XML Document](image)

The figure above is a DOM representation of an XML document containing the stock quotes information of two shares. The document has a root element called Quote, within this root element there are two children called Stock. Each represents a stock quote for a particular share. Each Stock element has two attributes, the first attribute has a name Symbol which stores the stock symbol, the second attribute has a name called price, which stores the price of this quote.

To create this document using the DOM framework we would write the following code:
To manipulate this document using the DOM framework, we would need to use our reference of the stock_quotes document and traverse through the hierarchy through the DOM tree to the required entity. For example, if in the above tree we would like to know the quote price of the stock with symbol “SUN” we would write the following code.

```java
Element stock2 = stock_quotes.getRootElement().getChildWithTag("Quote").getChildWithTag("Stock2");
stock2.getAttribute("Price");
```

The first line traverses through the tree to obtain the second leaf node. The second line obtains the value for the price of this stock entry.

### 4.6.4 Design Implications

The structure highlighted above provides some implication onto the underlying design of this toolkit. First of all, it is clear that each of the DOM entities including Documents, Element and Attributes are each separate objects that has their own internal states and methods to access their internal states.

Secondly, it is necessary some how to encapsulate this hierarchical arrangement of elements into our toolkit. The simplest way to achieve this is for each element to hold their own references to their children and attribute. Operations such as getChildElement() or getAttribute() would then become returning of these references back to the receiver.

### 4.6.5 Class Diagram

Bearing all the above considerations in mind, I have come up with the following design for XML creation and manipulation part of my toolkit (figure18).

Explanation of the purpose of each class is shown in the table below:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document</td>
<td>Represents a XML document</td>
</tr>
<tr>
<td>Entity</td>
<td>Abstract class that represents either a node or element in the</td>
</tr>
</tbody>
</table>
Information Collaboration on Mobile Networks
Fu Ting Chan

<table>
<thead>
<tr>
<th>Element</th>
<th>Represents an XML element.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>Name and value pair of information</td>
</tr>
<tr>
<td>ElementPath</td>
<td>Class which contains an Array of Element representing a path in the XML Document</td>
</tr>
<tr>
<td>Namespace</td>
<td>Name space information of document, to qualify and differentiate the tags of the element in the document.</td>
</tr>
<tr>
<td>KJDOMException</td>
<td>A kilobyte java DOM exception, can be raised when manipulating, encoding and parsing the XML document.</td>
</tr>
</tbody>
</table>

Figure 18: Design of XML manipulation model in toolkit
4.7 Binary XML Parser Design

My parser design, similar to the design of the WBXML encoder described in section 4.5, would use the visitor pattern to traverse the tree recursively in the width first fashion. The following is a class diagram showing the design of the WBXML parser.

![Class Diagram]

The abstract class `parse` defines an interface that all concrete parse must implement, in this case, the abstract methods are `parseElement(Element)` and `parseAttribute(Attribute)`. It’s concrete class `WBXMLBuilder` would be responsible for parsing a network input stream of bits that was encoded with the WBXML standard, once a whole serialised document is parsed, the method would construct a document using the DOM framework implementation that was designed in section 4.6 and return a XML document representative of what has been read in from the bit stream.

The WBXML parser should, similar to the encoder, not have to worry about the underlying implementations of the entity that is being parsed, instead the static method `parse` should be called to parse the appropriate element being considered.
Concrete implementations of the parsers could be extended through sub classing the parser class and implementing the parseElement and parseAttribute methods to extend the toolkit to provide other protocol of communications.

4.8 XML Document Serialisation

The CLDC (connection limited device configuration) package from Java 2 Micro edition provides a limited set of Stream classes for manipulation of network input output data independent of the underlying network implementation. The package includes the basic classes InputStream and OutputStream which are the class for representing an input and output stream of bytes. The concrete classes that our toolkit is most likely to use are ByteArrayInputStream and ByteArrayOutputStream. They provide an internal buffer of bytes that are read or sent to and from the network input and output streams. This means that our encode() and parse() methods that is in our WBXMLEncoder and WBXMLOutputter class should take as a parameter the abstract class InputStream and OutputStream to allow for the maximum flexibility when it is used. Within our toolkit we should use the classes ByteArrayInputStream and ByteArrayOutputStream to buffer intermediate results during encoding and parsing of the XML document. The following diagram illustrates this concept.

![Diagram of XML Document Serialisation]

Figure 20; Serialisation of XML Document

The diagram shows the design of the serialisation process of the XML document. First the document is encoded using the encoder into WBXML format, results of the encoded XML are then stored into a buffer which is used for transmission. On the other end of the transmission, the receiving device stores the intermediate array of bytes into a buffer which is then used for parsing, after which an XML document is produced from it’s binary form.

4.9 Platform Considerations

It is important during design to also take into account the platform that this toolkit would be running on. Appendix 12.8 shows a table of the different device profiles of wireless devices that could use this toolkit.
5 Part 1 - Implementation of Mobile Communication Toolkit

This chapter would describe the implementation of my mobile communication toolkit.

5.1 Implementation of DOM Tree

Implementation of element tree structure is relatively straightforward, the use of the CLDC Vector structure has catered for the storage of references of tree entities. Each element has two vector representing the contents and attributes that the element has respectively. Add and removal of attributes to an element are therefore translated to add and remove on the vector data structure with some error check code to ensure that the conditions are right before adding and removing of entities. For example when adding a child element to an element, we want to make sure that the child element does not already exist in the entries of contents, or else we would have double references to the same object.

5.2 Implementation of Encoder

5.2.1 String Table

A String Table in a binary XML document stores a table of strings that is being used in the document, including element tags and attribute names. Implementation of the string table relies on the use of a Hashtable to store a list of all the strings that the encoder has already encountered, and a buffer to write all the intermediate entries in the string table. The following procedure shows how this is done.

```java
private void writeStringTableRef(ByteArrayOutputStream buffer, ByteArrayOutputStream stringTableBuffer, Hashtable stringTable, String s, boolean tag) throws IOException{
    if(!tag)
        buffer.write(WBXML.STR_T);
    Integer index = (Integer) stringTable.get(s);
    if (index == null) {
        index = new Integer (stringTableBuffer.size ());
        stringTable.put (s, index);
        writeInlineString (stringTableBuffer, s, false);
        stringTableBuffer.write(WBXML.SWITCH_PAGE)
    } else {
        int offset = (int) index;
        buffer.write (WBXML.OFFSET, offset);
    }
}
```

The above procedure is taken from my WBXML encoder class, it allows encoding of the string s using the string table. In line 2 and 3 we check whether we are writing an element tag, if so, a special global token STR_T must be first written to the buffer. Line 4 checks whether there is already an entry of this string (ie if this string has appear before during the process of parsing), if this is not the case, we will need to add an entry of this string in the string table, which is done in line 6 to 9. This involves first finding the offset that this string should have from the beginning of the string table, then the string entry is added to the hashtable as well as the buffer. Finally in line 9 we write a byte the token SWITCH_PAGE which is a WBXML delimiter for this entry in the string table.

Since the string table always appears at the beginning of an WBXML document, we would need to complete the encoding of the whole document before we could start writing the string table to the output stream.
5.2.2 Global Tokens
The WBXML specification has defined a set of global tokens which are a predefined set of octets, I have decided that it is the best to separate the definition of these global tokens from the implementation of the encoder and parser by creating a new class called WBXML. This class has a set of constant definitions that can be accessed by both the encoder and the parser, so that we can make sure the same set of global token are used for both parsing and encoding the document. Also if we need to extend these token we would not have to change either the parser or the encoder.

5.2.3 Encoding Multi Byte Integer
The WBXML specification has specified that multi byte integer must be encoded in to a series of octets in such a way that the most significant bit of each octet acts as a flag to indicate whether it is the end of the multi-byte integer and the rest of the 7 bits encodes the value of the integer. I have approached implementing this using the following algorithm.

```
Procedure WriteInteger(int n)
    Holding list
    Loop through integer{
        Store the value n& 0x7f in holding list
        n = n>> 7 (shift n 7 bit to the right)
    }
    while holding list is not empty{
        obtain last element of holding list store it in temporary variable
        if(size of holding list >1)
            write to buffer the value (temporary variable | 0x80)
        else
            write temporary variable to buffer
        remove last element in holding list
    }
```

Essentially the first loop in the procedure split up the integer into group of seven bits and store these groups in a holding list with the first bit flag set to 0. In the second loop, we traverse through the holding list in reverse order as when it was put it, and for each individual byte we add a 1 to the most significant bit until we have reached the end bit in which case we leave the first bit as 0.

5.2.4 Encoding Elements
Element encoding utilises the WBXML global tokens LITERAL, LITERAL_A, LITERAL_C and LITERAL_AC. The presence of the letter A after the under scroll of these token signifies that this element has attributes. Similarly the presence of the letter C signifies that this element has a child element. When encoding an element, the encoder uses the DOM API to check whether an attribute or child exists, and adds the appropriate token before encoding each element according to the result.
5.2.5 Encoding Attributes

Attribute names are encoded as references to the string table, and attribute values are encoded as inline string, since there is little chance for a particular attribute value to appear more than one time in the string table, it is more appropriate to encode attribute values as inline strings helping to reduce the length of the string table.

5.3 Implementation of Parser

5.3.1 Parser State Machine

When parsing a series of tokens from the input stream, the WBXML parser maintains a state that is associated with the code space that the currently read token is in. The meaning of these tokens depends on the context in which it is used, i.e., depends on the state that this parser is in. The occurrence of the code page switch token (SWITCH_PAGE) while in a given state changes the current code page for that state. This code page will then remain as the current code page until another SWITCH_PAGE is encountered in the same document. The parser remains the current code page that it is in, initially this is initialized as 0. The following is a representation of the state machine.

![State Machine Diagram]

My parser implicitly maintains the current state when parsing an WBXML stream by traversing the current control between the readAttribute method and the readElement method. When the parser is current executing the readElement method in an element, we are in the Tag State, similarly when we have finished reading the tag and enters the readAttribute method, we are in the attribute state.

5.3.2 Parsing References

The string table is conveniently placed at the start of the document in the WBXML specification, this allows our parser to read the string table at the beginning. Entries of string are stored in a hashtable together with its offset in the string table acting as the key. When the parser subsequently encounters a reference in the document, it will lookup this reference from the hashtable and replace the appropriate name as the referenced value.

5.3.3 Parsing Elements

Elements are parsed using the encoded LITERAL token which is the first byte that the parser would read at the beginning of any element. The value of the LITERAL will act as an indication as to what to expect in this element. A LITERAL_AC token for example would tell the parser to expect to parse attributes and more elements after completing parsing the tag of this element.

5.3.4 Parsing Attributes

Parsing attributes involves reading a string table reference for the attribute name and a inline string for the attribute value.
5.4 Error Handling and Debug Functions

Error that are found during the manipulation of the toolkit are raised using a class called KJDOMException. Essentially this is a class that extends the java.lang.Exception class that process application developer a means of determining the type of exception that is being raised, allowing an indication to them as to what kind of problem there are. Within my toolkit, I have also added a lot of error checking code in different methods, once an runtime error is found, a KJDOMException is raised with a message indicating the kind of errors exists. Error checking codes includes maintaining the integrity of the references in the DOM tree, reporting of errors such as accessing a non-existence child or attribute or insertion of attributes with the same name to the same element. This would provide assistance for users to debug their applications at the same time maintaining in the Java framework of exception.

5.5 Encryption and Compression

The real challenge of incorporating the functionality of encryption and compression in my toolkit is to avoid tying my toolkit to a particular encryption or compression algorithm. This is because up to now, there is no standard technique to encrypt or decrypt an XML document, there are many different algorithm that could be used for textual or binary data encryption. Since encryption changes the underlying bit pattern that is being send and receive, we would effectively loose the application independency of our toolkit if we use a particular algorithm.

Therefore I have decided to abstract away the detail of encryption and compression from my toolkit. This is achieved by using the abstract InputStream and OutputStream class for encoding and parsing any data. Users can then use the secure input and output stream class to parse and encode an XML document, similar to the classes provided by the standard Java crypto package. Essentially my package use the bear minimum interface to send and receive data, and if application developers wants to use a secure or compressed stream they can pass this as a parameter into my encode and parse method to allow for such functionalities.

5.6 Hashcode() Function

In order to allow application developer to easily identify an XML document, I have implemented the hashcode() and equal() function in the document class. The hashcode function of a document returns an integer value that is dependant o the contents of a document. This is calculated by concatenation of the string representation of all the elements and attributes that is in this element and calculating the hashcode of this string. This means that two XML document which has exactly the same structure and contents would produce the same hashcode. This allows users to check whether two document’s content and structure is equivalent by compare the hashcode that each produces.

Another related function that I have implemented into the API is the sameStructure function, this checks that two XML document has the same elements with the same tags and attributes, but with different attribute values. This is sometimes useful to identify a particular type of document that has been received,
5.7 Implementation Challenges

5.7.1 Reference problem

One problem that I had whilst implementing this toolkit is to do with the situation when multiple elements has the same child. For example in the following code:

```
1. Element e1 = new Element("Parent1");
2. Element e2 = new Element("Parent2");
3. Element c = new Element("Child");
4. e1.addChildElement(c);
5. e2.addChildElement(c);
6. ....
7. e2.removeChildElement(child);
8. c.getParent() //error!
```

Line 8 would produce an exception because the addChildElement method in line 5 would set the parent reference of the element ‘c’ to ‘e2’, and once ‘e2’ has removed ‘c’ in line 7, element c would not have any parent, despite what had happened in line 4.

The underlying problem here is that addChildElement(c) in line 5 actually removes c from e1, without the user performing any operation on e1. This problem stems from the fact that we want to avoid an element having multiple parent which is semantically incorrect for an XML document tree.

There are a few solutions to this problem that I have considered below.

5.7.1.1 Forbid adding of a child that already belongs to a parent

This is the simplest approach to take, it forbids user adding any child element that already has a parent. ie we disallow line 5 above. This can easily be implemented by added an error check code in the method addChildElement, this is shown in the code shown below:

```
AddChildElement(Element child){
   if(child.getParent()!=null)
      throw new KJDOMException();
   .......
}
```

The above code checks whenever addChildElement is called, whether child’s parent variable is equal to null, if this is the case it means that the child has not yet got a parent. This method is easy to implement but does not provide the flexibility that we would want. For example we may create an element representing our name, and we would want to add it to all documents that we send out, we will have to create a new instance of an element representing our name each time and add it to every document that we send out, instead of having one instance of the element for all documents. We tackle this problem in the next approach.
5.7.1.2 Create multiple instances of the same child

This is the approach that I have taken in my toolkit. Basically whenever a new child element is added to a parent, and a parent already exist for this child, a new copy of this element is added, so essentially we have two instance of the same element each belonging to different parent. This means that the user can manipulate each of these children independently of each other which provides extra flexibility for application developers. Implementation of this approach is shown below.

```java
public Element addChildElement(Element child){
    if(child.getParent()!=null){
        Element newElement = child.copyElement();
        addChildElement(index, newElement);
        return newElement;
    }
}
```

Essentially the above code checks whether the child has a parent, if so, add a copy of the child to this parent, and return this new copy back to the receiver for further manipulation. The method copyElement is a method that I have implemented which returns an copy of the element to the receiver. The usual clone() method was not used because CLDC does not yet support the interface clonnable.

5.7.2 Printing and Error Tracing

Another challenge that I have found during the implementation of this toolkit is for tracing the exact source of the error during the development process. The problem roots from the fact that unlike in a workstation environment, on mobile devices, programs are executed without a text based console, so it is difficult to trace at which point did the program failed. But with the release of the KVM debugger in new development release this problem has disappeared. This development release of the KVM allows the user to specify an option to print all standard output to a buffer which can later be viewed. A screen shot of this is shown here.

I have also written a routine called toString() for elements and attributes to compliment this, allowing the printing of the full document object as string in a hierarchical fashion.
6 Part 2 - Design of XML Server

This chapter would describe my design of my XML server whose main function is to interface databases to mobile devices.

6.1 Design Decisions

This section would describe some of the design decisions that I have taken using the information from my background research and the nature of this project.

6.1.1 Choice of Environment

I have decided to develop my server in Java, this is because firstly of the fact that my mobile communication toolkit is developed in J2ME. Since J2ME is a subset of the Java 2 Standard Edition API, the toolkit would work without any modifications it’s code on a PC. By using Java on my server, I can easily achieve direct communication between my server and the toolkit without the need of any intermediate bridge. Another reason for using Java is because it provides the platform independence that we need, so that our server could run on Windows, Linux and other platforms without having to rewrite the whole application.

6.1.2 Choice of database server

I have decided to use MYSQL as the database to use in this server. This is because it is easy to install and administer and runs in different platforms. The fact that I am using a particular database in my development should not constrain on the portability of our server, this is because SQL is a specification nearly all SQL server complies with, the only thing that has to be changed in our server if we want to use a different database is to install a different JDBC driver to our server. JDBC is a standard for Java to database communication.

6.2 Use of Communication Toolkit

Central to the design of this server and the server configuration tool is the use the communication toolkit that I have implemented in part 1 of this project. It’s use can be mainly divided into two category.

6.2.1 Use of binary XML for communication

This server uses the toolkit to accept incoming requests from mobile clients using the WAP binary XML standard. A thread that is responsible for handling client request using the blocking receive function build(inputstream) in the class WBXMLBuilder to parse incoming requests for database operations. The toolkit is also used to sent the results of the queries back to the mobile clients, using the output(outputstream) function in WBXMLEncoder. This way we can conveniently send and receive information in a hierarchical format to mobile clients without having to deal with low level details of encoding, parsing and network IO streams that was dealt with in part 1 of this project.

6.2.2 Use of binary XML for configuration

Binary XML is also used for configuration in our server. Configurations are represented in XML format and consists of information such as the type of documents that this server is allowed to receive, the template mappings that tells a server what to do when a particular document is received, as well as
information about XML documents that this server can send. These ‘configuration documents’ are stored into a database in a binary format which can be modified using a server configuration tool. When the server starts up these configurations are loaded from the database into memory for quick lookup of template mappings.

6.2.3 Overview of how communication toolkit fits in XML server
The follow diagram shows the relation between my mobile communication toolkit and my XML server.

![Diagram showing the relation between mobile client, mobile communication toolkit, and XML server.](image)

Figure 21: Use of toolkit in XML server

The diagram shows that the toolkit is used to interface between mobile client communication and the XML server. If the client is using the same toolkit for communication a similar diagram would be drawn at the client side, but for the purpose of this chapter we abstract away details of the client, all we know is that the incoming request documents are encoded in WBXML format and the server’s job is to handle the incoming request and send a reply if necessary.

The second use of the interface is also shown in the diagram, we can see that the toolkit is used store configurations of the server in WBXML format.

The fact that the toolkit is used for communication or configuration should not make a difference to the implementation of the toolkit. We have seen in chapter 5 that the two classed WBXMLEncoder and WBXMLBuilder each would encode and parse a document comprehensively given an input stream and output stream. This stream can actually be a buffer than could be later written into either a database or an underlying network, the toolkit should not treat them differently, and hence provides it with the application independence that we aim to achieve.
6.3 Design of Server

The main purpose of the XML server is to ‘handle’ requests made by mobile devices. The way that the server ‘handles’ incoming request depends on it’s internal configurations mappings, which are defined dynamically at run time.

6.3.1 Server Architecture

The high level architecture of the server is given below.

The server uses the mobile communication toolkit which we have discussed in the sections 6.2. It also has a set of configuration mappings (drawn in broken line) which defines how to handle a particular request document. Lastly the server has database capability, which is what most mobile client is trying to access in their requests, but the framework of this server does not really limit it to this use only.

6.3.2 Start up Routine

At start up the server must run through a standard routine to make sure that everything is ready before accepting XML requests from mobile clients. This includes the following.

6.3.2.1 Socket Initialisation

The server must initialise a socket listening on the appropriate port number to accept incoming connections from clients. If the port number is not available for some reason, an appropriate error message must be used to notify the user.

6.3.2.2 Database connection initialisation

The server must initialise the database connection by loading the appropriate database driver and establishing a connection with user defined settings. This will be used for retrieval of settings and configuration information for the server.

6.3.2.3 Loading of configurations

Configurations of the server including all the template mapping must be retrieve from the database and loaded to memory using the database connection established.

6.3.2.4 Registration of user events handler
Handler of the text console and graphical user interface must be registered to allow user interaction with the server.

### 6.3.3 Request Handling

The server must be able to simultaneously handle multiple requests from different mobile clients. This means that our server must be multi threaded. Each client connection session may contain request that requires access to both the template mappings and database information, therefore it is necessary to allow each connection thread to access these resources in a thread safe manner. The following is my design of the requesting handling model in my server.

![Diagram](image)

Figure 22: Request handling model

The above diagram shows the situation of a mobile client establishing connection with the server, the connection request is indicated by the arrow labelled 1. The connection manager class in the server who is responsible of accepting incoming connections, would fork a new thread called request handler. This thread will then be responsible for handling of the client’s request and any subsequent request from the client until the client disconnects to close the session. Subsequent communication of this thread is with client is indicated by the arrow 2 and 3.

It is important to note that for every handler thread created, a new database connection is created with the handler, this is indicated by the circle with the broken line in the handler. This is because nearly all client request would require database connectivity and this is provided with the handler itself.

I have also considered a different architecture whereby we have a pool of database connections that are shared amongst handler threads and handler threads request for a database connection when required. This is a more efficient design because it will almost certainly require less database connections to be created as not every single handler thread would be performing a database operation at any one time.
However having considered the possibility of allowing mobile clients to retrieve data from different databases, it is potentially more extendible to allow every handler thread to manage their own database connection. By doing this the handler thread can read in requests from the client and make database connection that the client requires, rather than having a pool of connection threads that is the same for all handler threads. This allows greater flexibility in our system.

Another design implication that has become apparent in this diagram is the use of a single template map for all handler threads. This means that we have to take care to ensure that the modification of any state of the template map is serialised, i.e., it occurs as a monolithic operation. Also to ensure that deadlock or livelock doesn’t exist in our system. Generally this would not be a problem in this design because we would not be modifying any state in the template map, instead all states are encapsulated within the handler thread itself. So this could ensure that our server is thread safe.

6.3.4 Template Map Arrangement

When the server starts up and loads the entire configuration from the database into memory, each template in the configuration runs through an re-arrangement process to allow template mappings to be stored in memory in an efficient manner so that subsequent access could be quickly made to these mappings. The following diagram shows how this arrangement is made in memory.

![Figure 23: Arrangement of configuration in memory](image)

Each Configuration is stored in memory and accessed using the Configuration class. As well as loading of a configuration, this class is responsible for holding the reference to the first node of the configuration. This is the input XML document that represents a request that can trigger the chain of actions that is defined in this configuration. The input document class holds reference to the next action
node in the chain. An action node represents an XML document that defines a particular action that the server has to perform. Actions includes, database operations, branching conditionals, sending an XML document out and lastly an XML document that can be received.

6.3.5 Template Map Lookup

When an incoming document has been received from a mobile client, the request handling thread has to lookup it’s list of loaded configuration in memory to see which one has an input document that is the same as the request. This is achieved by using a ConfigMap class that is responsible to keeping a map of all the configurations in the server. This lookup function would then become looking through all the configurations in this server and calling the sameStructure() method which is defined in the element class of the toolkit designed in part 1. the sameStructure() function checks whether two XML element has the same structure without referring to the values of each element. This is important since we cannot define every possible XML document instance that a server can receive, all we to do is to make sure that the structure of the require and the stored template of input document is the same.

6.3.6 Server Interaction

I have designed the server to allow both command based and graphical based user interaction.

6.3.6.1 Text based Console

The text based console would allow user to enter command lines in the terminal window where the XML server was started. A separate thread called console listener constantly reads in user commands in the terminal window and perform the requested function.

6.3.6.2 Graphical User Interface

The server has a GUI built using Java Swing classes to provide users with a graphical interface to manipulate and access information about the server.

6.3.7 Output logging

A class called OutputLog will be implemented to allow writing of access information and server status to a file.

6.3.8 Fitting it all together

Taking all of the above design considerations into account, the following is my design of the XML server. Class diagram shown in figure 24.
Figure 24: Class diagram of XML server

The design of the server mainly reflects four aspects that I have considered, namely request handling, configuration mapping, output logging, and user interfacing. A summary of the purpose of each class is as follows.

**ISyncServer** – class that contains the main method and initialises all other classes, also contains references to configuration maps.

(request handling)

**ConnectionManager** – loops continuously to accept new incoming mobile client connection, hands control over to handle connection once a connection is established.

**HandleConnection** – Responsible for server a client requests continuously until client disconnects

(Configuration Mapping)

**ConfigMapping** contains all the configurations there is in this server, also responsible lookup of input documents that trigger configuration actions.
Information Collaboration on Mobile Networks
Fu Ting Chan

Configuration - Responsible for loading and storage of individual configurations.
ActionNode - Represents an elementary action that can be performed, links together to form configuration chain

(output logging)
OutputLog - Gives a convenient way for all other classes to write information to the log file.

(user interface)
ServerConsole - Graphical interface for server manipulation
InfoDialog - Dialog that displays a given string message to the user as a pop up dialog.
ConfigDialog - Dialog to view configuration that has been loaded to the server
ConsoleListener - Command based interface that allows users to manipulate the server.

6.4 Design of Configuration Tool
The main purpose of the configuration tool is to define a set of template operations that determines how the server would handle request documents from mobile clients.

6.4.1 User Interface
The user interface design of the configuration tool consists of a series of numbered tab menus, listing in order the processes that the application developer have to follow in order to define a template mapping for the server. There will be four tabs, for database connection, legal document definition, schema editor and process mapping designer.

6.4.2 Database Connection
The configuration tool uses the database as a mean of storage of all configuration information. It is also where the server would ultimately load the configuration information from. So therefore one of the first thing the user have to do after starting up the configuration tool is to connect to a database server. To do this, our tool will have to load the JDBC driver for the database server and establish a connection given information such as server host, user name, password, port number and database name.

6.4.3 Legal document Definition
This allows users to define the structure of the XML documents that the server is allowed to receive. The user should be presented with a wizard interface to follow to input information such as document title and document description. The wizard should also allow the user to define a hierarchical tree of the structure the XML document to receive. User should be able to specific the tag name of each element in the document as well as attribute name and values of all attributes including the data type to expect from a particular attribute value. The configuration should also support multiple attribute values within an attribute, to allow more efficient use of XML documents for requests. Definitions of these document structure are stored into a database. The wizard also provides users with the ability to edit and delete these definitions from the database.
6.4.4 Database Schema Designer  
This should provide the user with a wizard to edit the schema of the connected database. This is necessary because application developers might need to extend the database to include some extra information that the database has to accommodate for the mobile applications. This tool provides application developer a means of doing this using SQL commands.

Since there are a lot of schema editors which comes with distributions of databases, I have only decided to include an editor with minimal functionality here. Application developers are free to use another other schema editor to extend the database.

The editor should provide user with a box to type SQL commands, it should also have a message box to hold error messages that are created from executing SQL queries. I have decided to display messages that are created by the JDBC database driver and display this directly to the user. This could be done because JDBC driver raises exception when a particular SQL statement is malformed or if there are problems executing it, all I will have to do is to display the textual representation of this exception, this would provide the developer with a fair idea of where the problem is.

6.4.5 Design of Process Mapping Designer and Drag and Drop interface  
This is the most important step of configuration definition. It allows developers to construct a sequence of actions that the server performs when it receives a particular XML document. I have included four basic building blocks of actions including receive XML document, send XML document perform a database operation and branch on conditional. Full details about each building blocks including their XML definitions could be found in section 0. The idea is to allow the user to construct a sequence of actions by connecting these building blocks together with connectors through a drag and drop interface. So user can drag a block representing a receive XML document action, then connect this block with subsequent blocks forming a flow graph of actions. The flow graph will then be a graphical representation of the operations to perform when a particular XML document to receive and the user can easily manipulate connections between these blocks to change the underlying logic behind the server.

As an example, the following flow graph represents the configuration for handling search request for stock quotes prices in the scenario discussed at section 3.3.2.2.
The above is what the flow graph the user constructs for this purpose should look like. The rectangles represents actions nodes, ie to request for the server to perform a given operation. Action nodes has input and outputs represented by the little circles that is on the nodes, if a circle lies on the top horizontal line of the rectangle, it signifies that this is an input of the action node, similarly if the circle lies on the bottom horizontal line of the rectangle, this signifies the output for the action node.

The straight line that links the rectangles are connectors, they are connected to the input and output of each action node and represents data and control transfer from output of one node to the input of another node.

So figure 25 represents a template mapping that says:

1. (first action node from top) If the server receives an input document with tag stock_quotes, accept request, and initiate the following chain of action.
2. Extract the attribute value of symbol from this request, and perform a select operation on the database. (The text in bold illustrates an example).
3. Now perform a conditional on the result of the database operation, if the input (of this node) with field name “price” is empty, then send an XML document with no_quote_found back to client Else return results of query in an XML document called quote to client.
Using this drag and drop method of constructing flow graph allows application programmers to construct back end operations without writing any code at all. All they have to do is to connect different action nodes together to build a flow graph that defines dynamically the behaviour of the server.

To demonstrate the power of this concept, let’s say the application developer now decides to extend the above template to include the extra functionality of notifying another user if the price of the share in question is lower than 50 say. The application programmer have to do is to open up the template designer again, and do the following.

```
if Symbol = SUN
  SELECT price FROM stock_table
  where SYMBOL = "SUN"
```

```
Database operation
price="65"
```

```
if price<50 then
  else
    Conditional
```

```
if price isEmpty then
  else
    Conditional
```

```
XML OutputDocument (stock_alert)
  destination = x
```

```
XML OutputDocument (no_quote_found)
```

```
XML OutputDocument (quote)
```

Figure 26: Extension of stock_quotes

The user has added an extra conditional to check whether the price is less than 50 pounds, if so send an output document to user at destination x with the root element containing a tag called stock_alert and contents of stock prices in the document. Then continue replying client. The example demonstrates how the behaviour of the server can be easily modified without having to change any code of the server.
Template definitions are stored into the database when user saves a configuration, it is stored as a big XML document with child elements representing the action nodes and connectors. Definition of these are detailed in section 6.5.

6.4.6 Putting it all together

Having discussed the different functionality above, the following is a class diagram showing the design of the server configuration tool.

The following is a brief description of the functionality of each of the classes.

**DynamicConfig** - This is the class that contains the main user interface, with 4 tab menu to navigate to

**JTabPane** - These 4 tab panes contains interface for database connection, WBXML editor, Schema Editor and process mapping designer.

**WBXML_EDIT** - Wizard interface for definition of XML document structure that server can receive, uses a tree interface in Swing. Jtree and JNode

**DB_EDIT** - Wizard interface that allows editing of database schema

**Process_Designer** - A drag and drop interface that allows user to construct template definitions using the building blocks documentNode. Condition node and DBOperationNode.

**ProcessNode** - Represents a node in the user interface, defines the general behaviour of a node, how it interacts when mouse clicked or dragged. This way class that extends this class would inherit the drag and drop behaviour. An abstract class is also defined, called openEditPanel, which must be implemented by concrete subclasses, it represents a method to open a dialog for editing of the details of the node. The method is called when user press a button in process_designer. Process node has a document variable,
which is an XML document representation of the current node, this is used in designer
when user wants to save configuration to database.

**DocumentNode** - Represents an XML that the server receives or sends, user can open up edit panel to
define whether the document is for send or receive. Has an abstract method called
XMLDocument_Init that makes sure that it’s subclasses implements, the method is
used to initialise the node document object of the two subclasses, because their XML
document representation would have different structure.

**InputDocumentNode** - This represents a document that is received. Implements the XMLDocument_Init
function and writes an XML document of the current node settings to the variable
nodeDoc.

**OutputDocumentNode** - A document that could be sent out from this server. NodDoc includes mapping of
output element values to input of current node. Has a dialog called
OutputDocumentNode_define to graphically define settings of this node.

**ConditionNode** - Represents a conditional, implements the method openEditPanel, to open panel to
define details of the conditional to branch on. Edit dialog is called
conditionNode_define.

**DatabaseOperationNode** - Represents a database query operation. Implements the method openEditPanel, to open
a dialog that defines the type of database operation, including SELECT, INSERT,
UPDATE and DELETE. As well other information for the operation. Edit dialog is
called databaseOperationNode_define.

### 6.5 XML Configuration Definition

There are four type elementary actions that a server could perform. Information about each of these
actions could be represented as XML documents, in this project I have referred to these documents as action
nodes. This section would describe the purpose of each action node as well as examples of their
representation in XML. The reason why these documents are represented as XML documents is because this
way they would be stored as configurations in the database, this way the server could be notified of the
updates to the configurations and load the new settings as XML configurations from the database.

### 6.5.1 Input Document

This represents an input XML document that the server could receive. The following is an
example of it’s representation in the server.

```xml
<XML_Document Name=LOGIN_REQUEST Type=INPUT DocId=15 nodeId=0
  root_element=login_request >
  <Trigger_Element>
    <login_request username=VALUE DATATYPE=String password=VALUE
      DATATYPE=String />
  </login_request>
  <XY_Coordinates X=182 Y=67 />
  <Input Name1=login_request::username DATATYPE1=String Name2=login_request::password DATATYPE2=String />
  <Output Name1=login_request::username DATATYPE1=String Name2=login_request::password DATATYPE2=String />
</XML_Document>
```
The root element XML_DOCUMENT identifies that this document is an XML document. The attribute type has a value equals to INPUT which tells the server that this is an input document. Here we can see that this particular input document has a name called LOGIN_REQUEST. The structure of the document is defined in the trigger_element tag, so for this configuration to be triggered, we need a document which has a root element with tag login_request with two attributes, the first one is username which is of data type string and the second attribute to expect is password which also is of type string.

The document also has information regarding this node’s input and output, defined in the last two sub-elements.

### 6.5.2 Output Document

An output document is a document that this server can send out. The structure of this document is very similar to the input document, apart from the fact that an additional child element called output_mapping is added to the root element. Essentially this sub element tells the server what content to fill in for each of the elements when the server is about to send this document away. For example in the following fragment taken from a output document configuration, says that the output element should have the attribute located at the element login_reply to have authenticated set to true.

```xml
<Output_Mapping>
  <Entry login_reply::authenticated=True />
</Output_Mapping>
```

### 6.5.3 Database Operations

Database operations can take the form of either Select, Insert, Update or Delete. Here is an example.

```xml
<Database_Operation Name=New_Operation Operator=SELECT Table=login
    nodeId=2 MODE=ALL >
  <Column_Include Col1=username DATA_TYPE1=CHAR[] Col2=password
      DATA_TYPE2=CHAR[] />
  </Column_Include>
  <Where_Clause>
    <Entry WHERE_OPERATOR=AND Column=username QUANTIFIER==INPUT_NAME=login_request::username />
    <Entry WHERE_OPERATOR=AND Column=password QUANTIFIER==INPUT_NAME=login_request::password />
  </Where_Clause>
  <XY_Coordinates X=196 Y=281 />
  <Input Name1=login_request::username DATATYPE1=String Name2=login_request::password DATATYPE2=String />
  <Output Name1=username DATATYPE1=CHAR[] Name2=password DATATYPE2=CHAR[] />
</Database_Operation>
```
In the above example, the XML document defines a SELECT operation on the table login. A child element called column include defines which columns in the database to include. Another child element called where_clause determines the constrains that has to be taken into account when performing the SELECT operation. The document also contains a set of other child that represents input, output data as well as their x and y location in the drag and drop interface.

### 6.5.4 Document Condition

This represents a condition branching node where user can define the path of actions to be taken given a set of conditions. The following example illustrates a condition for checking whether the name and password value if of a particular value. Each conditional can have a set of conditions that the server branches on each represented as a condition_entry in the document.

```xml
<Document_Condition Name=Conditional nodeId=3 If_Condition=1 Else_Condition=4 >
  <Condition_Entry AndOr=AND Is=IS Field_Name=username Operator== Value=ftc />
  <Condition_Entry AndOr=AND Is=IS Field_Name=password Operator=< Value=project />
  <XY_Coordinates X=238 Y=512 />
  <Input Name1=username DATATYPE1=CHAR[] Name2=password DATATYPE2=CHAR[] />
  <Output Name1=username DATATYPE1=CHAR[] Name2=password DATATYPE2=CHAR[] />
</Document_Condition>
```
7 Part 2 - Implementation of XML Server

This chapter would detail the implementation of the XML server and its configuration toolkit.

7.1 Server Implementation

7.1.1 Multi threading in Server

I have used multi-threading in many different areas of my server, to ensure the server’s reactivity that is important for low response time to client’s requests. Threads are used to handle multi client requests, it is also used to provide the user a feedback of the progress of server’s start up, it is used for graphical and command based interfacing, it is also used for output logging. There are two main ways that I used to perform multi threading in my server, the first way is to extend the class thread, the second way is to use a class called the Swing worker which is an abstract class that could be extended to perform GUI related worked in a dedicated thread.

7.1.2 Request Handling

In order to achieve the behaviour of multi threading for handling of multiple simultaneous request from mobile client, the Class thread in the Java API is used. Essentially, we have the connection manager class which extends thread, the class overrides the abstract method run() which defines what to do when the thread is started. Within the run method, the connection manager loops continuously to accept an incoming socket connection. The follow code shows the code the core of this run method. When an incoming connection is detected using the serverSocket.accept() method, which is a method that would block until an incoming connection arrives, the server would first assign a number for the new thread that will handle this client (line 2). It will then pass this connection to a handle connection class (line 3) and insert a reference to this handler thread into a hashtable with the threadN number as its key, so that later on, we can reference back to this handler thread. The last line of code starts the handler thread.

```
theConnection = serverSocket.accept();
Integer threadNo = assignThreadNumber();
HandleConnection palmConnect = new HandleConnection(theConnection, server, this, threadNo);
handlerThreads.put(threadNo, palmConnect);
palmConnect.start();
```

Handle connection is another class which also extends thread, when started would be prepared to parse any WBXML documents that appears in its input stream, i.e., it is prepared to receive a request document. Handle connection is also responsible for looking up the list of configurations there are each time a request is received and determines the actions to perform with the request. One restriction applies when the handler thread accesses the list of configurations in the server, it is that these handler threads is not allowed to modify any state in the configurations which will corrupt the configurations that is in the server. This should not limit the functionality of our handler however, since all it really needs is to be able to read the configurations for handling requests.
7.1.3 Loading of Templates from database

I have implemented an interface that allows user to start up the server and load the list of configurations that exists in the database. The following is a screen shot of the interface.

The interfaces uses two separate threads, one for loading of the list of configurations from the database and one for updating of the user interface components to ensure that the correct information is shown to the user. The progress bar for example shows in percentage terms how much of the configurations were loaded. The scrollable text pane on the bottom shows the textual representation of the documents that has been read from the database. Here I will discuss in more detail how the two threads work together in synchrony.

Firstly how I actually retrieve XML documents from the database. XML documents representing configurations are stored in a table in a database using the WAP binary XML format. The database uses the data type BLOB to hold this document, and each document is referenced by it’s configuration ID. When the user starts up the server and presses the load configuration button in the user interface, the server would create a new thread that is responsible of reading the configurations from the database in the table using a SELECT query. Then the thread would continue onto building a document object from the input stream using the WBXML builder class in the toolkit. So conceptually this first thread performs the following task.

Database

\[
\begin{array}{c}
\text{binary documents} \\
\text{JDBC} \\
\text{binary results} \\
\text{WBXMLBuilder} \\
\text{document objects} \\
\text{Rearrangement} \\
\text{rearranged document objects} \\
\text{configurations}
\end{array}
\]

So the JDBC pulls the binary document data from the database, and passes it’s result to the WBXML build which then builds a document object that would be passed to another process for efficient rearrangement before it is stored as configurations for access by mobile clients.

The purpose of the second thread is to update the user interface continuously to display the progress of the first thread described above. Essentially this thread uses a timer, which ‘ticks’ every a tenth of a second. Whenever the thread ‘ticks’ it updates the user interface components to reflect the new progress in the progress bar, as well as displaying the messages the are written to a string buffer by the first thread.
7.1.4 Graphical Interface

The design of the graphical interface for the server uses a series of Java Swing components to provide an easy to use method of interaction from the user. A screen shot is shown below.

![Screen shot of main user interface](image)

**Figure 28: Screen shot of main user interface**

The server has a set of menus on the top that allows user to perform operations such as starting up the server, shutting down the server, viewing of the list of configurations that is available in the server and so on. In the middle, we have a panel which is called thread view that allows user to monitor any threads that are active in the server at the time. The two buttons labelled < and > allows user to navigate between different active threads, and the two scrollable text windows shows the current activity of each thread. We can see here that thread number 1 has connected to a client, and it has received an XML object called login_request, with attribute username equal to ftc97 and password project. The window then continues displaying the sequence of actions that is performed using this request, including finding of configurations that matches this request, and performing a database select operation on the table users.

Towards the bottom of the interface we have another text window that displays the status of the server.

In terms of implementation of this interface, the interface components such as menus, text box, buttons and scroll panes all uses the Java components that is provided with Swing. The thread view text boxes which shows activity of the active threads is implemented by holding allowing the server console
to hold a hash table maps thread number to a string buffer. Each newly created thread is assigned a
unique thread number within the server, and whenever a thread prints a message, the message is
automatically appended to the correct string buffer by the server console. This means that displaying
these thread messages would involve only a lookup in the hashtable of the appropriate thread number and
displaying the contents of the retrieved of the corresponding string buffer into the text windows.

7.1.5 Text console

The text console in my server is implemented as a separate thread called ConsoleListener which
constantly listens to what user types into the command line and stores the typed character into a buffer of
size 50 charters long. Whenever a carriage return (enter key) character is detect, the thread matches the
string that is received in the buffer with any predefined commands that exists, if a match is found,
perform the appropriate task and continue reading user input again. If no match was found, report to user
in text console.

7.1.6 Output logging

The OutputLog class is a simple class that allows other classes within the server to print text
messages to a file. It extends the BufferedWriter class which defines the basic framework for writing an
array of bytes into a location. The class has one public method write, which writes lines of characters
into a file.
7.2 Configuration Tool Implementation

7.2.1 Main Interface design

The following is a screen shot of the main interface of this configuration tool.

The main interface consists of four menu tabs, and is ordered according to the way application developers defines a configuration. The first tab is database connection, it allows the user to define settings of the database they wish to connect to.

The second menu is Edit WBXML document which allows the user to choose define the structure of XML documents that the server is allowed to send and received. When the tab is pressed a list of currently defined documents are listed with a short description of each document. Screen shot shown below:
The user can then choose from the list of documents to edit or delete from or to add a new document definition. The list of document title, document id and descriptions are retrieved together with other information automatically when user connects to a database in tab menu 1. New document definitions are stored into the database using the same database connection.

The third tab menu allows the user to launch a wizard to edit the database schema and the fourth tab menu allows the user to launch a template designer to define the behaviour of the server.

### 7.2.2 Legal Documents Definition

The WBXML editor wizard is constructed by chaining three JFrame components together. These are classes with identifier, WBXML_EDIT1, WBXML_EDIT2 and WBXML_EDIT3. These components are arranged in the following way.

![Diagram showing the chaining of components](image)

The arrows mean that a component has reference to another component. Using this ‘chaining’ method, a user can navigate through and back the wizard by using the references to each of the three components. The method setVisible(Boolean) is used to show and hide each step to ensure that the user can only interact with one of the dialogs at any one time.

A screen shot of the dialog WBXML_EDIT1 is shown below.

![Screen shot of WBXML_EDIT1 dialog](image)

The first step of the wizard allows user to enter a name for the definition and specify the tag of the root element of the XML document to send or receive. User can then press “next” to proceed to the next step. User can cancel the definition process at any time by pressing the cancel button.

Similarly WBXML_EDIT2 allows user to enter a description for the document definition. The final step WBXML_EDIT3 allows user to graphically define the structure of the document to receive. A screen shot is given below.
The interface consists of a main window that contains a tree representation of the XML document. In
the screen shot, we can see that the document consists of a root element with tag stock_quote, which has
two children, one is date and another is markets. The child market has further 3 children, being “DOW”,
“NASDAQ” and “FTSE”. User could expand and collapse the tree view by pressing the magnifying
glass that can be seen in next to the market node here. On the left we have a table called attributes which
shows a list of the currently defined attribute and data type of the attribute value for the selected node in
the tree on the left.

User can modify the tree by pressing the create element and delete element button whilst selecting a
node in the tree window. Create element button creates a child node for the selected node, the user then
have to type the tag name for this element. Delete element would delete the select node and it’s child
nodes.

User can also modify the attributes that an XML document can have using the create, rename and
delete attribute button on the right. I have also build right click menus that allows user to conveniently
manipulate the tree, when the user right clicks the mouse in the tree window, a menu would pop up in the
location of where the user clicked. An example of this can also be found in the screen shot.

In terms of implementation this last dialog is the most complicated one out of the three. I will discuss
here how I have implemented the tree interface and the attribute table on the right.

### 7.2.2.1 Tree Interface

The tree interface is implemented by extended the Jtree class in the Java API. The following
diagrams shows how I have implemented this tree.
The tree is contained within a scroll pane, this ensures that if the size of the tree is larger than the window size then a scroll bar is created to allow scrolling of the current view port around the tree. Each element in the tree is a DefaultMutableTreeNode which is a concrete implementation of the class tree node. We also have a to create a tree model which tells the tree the model of information that is to be expected from the tree node. In order to display the tree we also need a tree cell render which draws the actual tree on the screen. The selection model of the tree defines behaviour of how the tree could be selected, for example whether to allow multi selection of nodes in the tree (in this case we define only single selection mode because we only want to edit one node at any one time), and the expand and collapse behaviour of the tree. The tree cell editor which is registered to the tree allow us to build in double click editing of nodes behaviour to the tree. Lastly the mouse listener is used to respond to the right click pop up menu behaviour of the tree.

### 7.2.2 Attribute Table

The attribute table on the right of the interface is built using the Jtable interface. This requires us to define in a similar fashion to the Jtree the table models, selection model, cell renderer, cell editor and mouse listener for the table instance. Since each cell of a Jtable can contain an interface component, I have decided to add a combobox within the cells of the last column “data type” so that user can choose from a list of valid data type that an attribute can take without having to type it in manually. A screen shot is shown below:
We can see that with each attribute, when a user clicks onto the data type cell, a combobox with all the valid data type pops up. This is implemented by setting the cell editor for the last column of the table to be a DefaultCellEditor that has embedded within it a JComboBox instance representative of the different datatypes.

7.2.3 Process Designer

The process designer is a drag and drop interface that allows users to construct a flow chart that represents the sequence of actions the server have to perform when it receives an XML document. As describe in my design in section 6.4.5, there are four building blocks for constructing the flow graph, these building blocks are referred as action nodes. All four of these action nodes inherits from a common super class called process node which defines the basic behaviour for all action nodes.

7.2.3.1 Process Nodes

The common behaviours that the process node class defines all action nodes shares includes the following:

<table>
<thead>
<tr>
<th>Action</th>
<th>Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drag and drop</td>
<td>Moves an action node from one point to another</td>
</tr>
<tr>
<td>Node Creation</td>
<td>Creates a new node, opens up a dialog if required to initialise node settings</td>
</tr>
<tr>
<td>Node Edition</td>
<td>Opens up a dialog for specifying details of the node</td>
</tr>
<tr>
<td>Node Delete</td>
<td>Deletes the node from the interface</td>
</tr>
<tr>
<td>Input</td>
<td>Defines the behaviour of the list of input data to this node</td>
</tr>
<tr>
<td>Output</td>
<td>Defines the outputs of this action node to the next</td>
</tr>
<tr>
<td>Connect Input</td>
<td>When input is connected, may open up node edition dialog</td>
</tr>
<tr>
<td>Connect Output</td>
<td>Update input of subsequent node in the chain</td>
</tr>
<tr>
<td>Disconnect Input</td>
<td>Update the output of current node</td>
</tr>
<tr>
<td>Disconnection Output</td>
<td>Update the input of subsequent node</td>
</tr>
</tbody>
</table>

These common behaviours define each action node’s basic interaction with the user. Process node also defines a Document object which is an XML document that represents this node, common elements
in the document of all action nodes includes definitions of X and Y coordinates of this node as well as information about the input and outputs of current node. Subclasses of process nodes is then responsible for filling in action nodes specific details of the document.

### 7.2.3.2 Input XML document

When user drops an document object onto the interface, the following dialog appears.

![Input Document Dialog](image1.png)

The table on the left shows the list of XML documents definitions there are for this server. The user can then choose between input or output document.

An input document represents an XML document that triggers this configuration. Shown below is the drag and drop icon that represents an input document. When a user adds this node to the configuration, this means that if this document is received by the server, and perform the chain of actions that is connected to this node.

![Input Document Icon](image2.png)

The upper white circle represents the input and the lower white circle represents the output of this node. The name of the document is also included with the icon of this input document.

Implementation of the input document node is not too difficult, most functionalities are inherited from process nodes, however it is necessary to define custom output procedure to make sure than when the output of this node is connected, it represents the different request parameters than this XML document can contain.

### 7.2.3.3 Output XML Document

This represents an XML document that could be sent out from this server. It’s graphical representation in the interface would similar to the input document apart from the fact that it’s input is connected and output is empty. Again this node inherits from it’s super class process nodes most features. We have to additionally include the behaviour of mapping input of this node to attribute values of the document to send out. This allows the server to fill in contents of this output document at run time. To do this, the output document node
must look into the attribute values in its document and construct in interface that allows the user to assign an input from previous step to this attribute value. A example of this is shown below.

On the left hand side we have an attribute called authenticated which is located in the login_reply element, and before we send out this XML document, we need to assign this attribute to a value which is designed on the right hand side. The tool also checks for the data type of the two variables to make sure that the assignment is valid.

7.2.3.4 Database operation

This node represents an operation that could be performed on the database. When a user drops a database operation node to the interface, the following dialog would appear.

This dialog allows user to choose define details about the database operation to perform. Here we can see that we have defined a Select operation on the table login in the database. And we want to include columns username and password into the output of this node. Also we define the ‘where’
Implementation of this dialog consists of mainly using the standard Swing components, a more tricky part was the where part of the statement, because it is not known in advance how many clauses there are in WHERE, so it is necessary to allow user to add and delete clauses using the + and – buttons. Each clause is a group of Java components which gets added and deleted together.

The problem of not knowing how many clauses there are in the WHERE part means that there can be arbitrary number of components in the interface, but when user press finish, it is necessary to read in all the values of each interface and store the into the definitions of the nodes. If we do not know at compile time how many user interface components there are, it will be difficult to read these values. My approach to this problem is to create a class that contains a group of components that represents the clause, and this class is responsible for reading and outputting the values selected for each of it’s components to the node. The XML toolkit was used to achieve this. The following diagram illustrates this concept.

```
finish(Element parent){
  Element clause = new Element("where_clause");
  clause.addAttribute("field", fieldBox.getText);
  clause.addAttribute("operator", operatorChoice.getText);
  clause.addAttribute("input", inputBox.getText);
  parent.addChildElement(clause);
  nextClause.finish(parent);
}
```

So within the ‘WHERE’ part of each statement we have an arbitrary number of clauses, I have decided to set up a chain of clauses in memory, so that each clause has reference to it’s next clause. When the user press ‘finish’ in the user interface, the first clause will be notified to save it’s data, this would run the code that indicated in the box on the right. The routine ‘finish’ would take in as a parameter the parent element. It would then add a child element with a tag name of “where_clause” and attributes defined as the state of each of the user interface components within the clause. After this operation has been completed, it calls the next clause in the chain to do this same. This way the element
parent that is passed in would have a list of children that represents the clauses of this `WHERE` statement.

The database operation node icon is as follows. The icon also shows the identifier of the database operation. In this case, it is called ‘find_user’.

![Database operation node icon](image)

### 7.2.3.5 Conditional Node

A conditional node represents a branching condition in the set of processes. When user drop a conditional node into the flowchart interface, a dialog appears to allow user to define what conditions to branch on. Each condition is a logical statement testing on the value of the input to this condition. There can be up to two output connections of a conditional node, one for the ‘if’ branch of actions and one for the ‘else’ branch of action. The following is how a conditional node is represented in the interface.

![Conditional node representation](image)

We can see here that the ‘if’ and ‘else’ branch is indicated on top of the middle node of the output connectors.

### 7.2.3.6 Connectors

#### 7.2.3.6.1 Interface

Connectors allows user to connect the input of an action node to an output of another action node. A connector consists of three circles.

![Connector interface](image)

The two ends of circles represents connection points to action nodes, whilst the middle circle is used to insertion of intermediate action nodes and for allowing the display for additional information about the connector branch, including the addition of “if”, and “else” words described previously. When an end point on a connector is connected to an action node, this end point changes it’s colour from white to red, to indicate that a connection is established. Similarly if a connector is removed from an action node, the circle would turn back to white.
7.2.3.6.2 Details of Implementation

When implementing the connector, I have found that there is no readily available swing components that is similar to a connector that I am trying to implement. So I decided to implement a custom component that represents this connector by extending the Class JPanel. The reason why I did not extend the JComponent class is because JComponent class does not provide with the appropriate refreshing functionality properly for the implementing a graphical component, and it was detailed on Sun’s website that by extending JPanel, the component would repaint properly when an UPDATEUI() method is called on that component every time the component is re-drawn.

As mentioned, a connector consists of three circles, and a line has to constantly connect between each end of the circle to the middle node. My method of displaying these circles within my connector is to use the Swing class JLabel. When the connector is initialised, it first creates three instances of the JLabel representing the circles. The next thing that it does is to register all the drag and drop handlers for each of these JLabel so that these circle will respond to mouse dragged events by the user.

As well as displaying the circles, I also have to ensure that the lines between the circles are drawn continuously when a circle has been moved. I accomplished this by overriding the paintComponent(Graphics g) method in my connector class. This method is called automatically whenever the parent component of this interface calls the repaint(graphics g) method. I added the code to draw lines between the circles within this method, therefore whenever a user drag and drops any of the circles within this connector, the code to redraw the lines are called. The following diagram explains the method more clearly.

The diagram illustrates what happens when a user drags one of the circles within the connector object in the user interface. When the top circle is dragged along the user interface, it triggers off events that would cause the container of this connector to repaint this area of the screen. Since the container knows that within this rectangle area, the connector component exists, it would call it’s paintComponent(g) method to repaint this component. This method which I overridden would execute the code to repaint the line between the circles. Moreover the parent would also repaint this area so that old lines are erased from the screen and only the new line is present. This avoids having a trail of lines obscuring the scene.

The advantages of extending the JPanel class to implement the connector is that I inherit the repaint functionality automatically without having to worry about when it will be called. If I didn’t inherit this function I would have to catch the mouse drag operation of each of these circles, find out which connector the circle is associated with, redraw the
lines of that connector and lastly determine which part of the screen is affected, repaint that part of the screen. As well as repainting, by extending JPanel, and hence being a JComponent, it makes the connector to be compatible with all the other user interface components that it has to work with. For example the scroll pane where it is added. If the scroll pane does not identify the existence of a JComponent in a particular view port, the component would not be painted after scrolling to and back of the pane.

It is also worth mentioning here how I implemented the method of for connecting a connector to an action node. Essentially, each of these circle components are registered to the mouse_released event, when a particular circle is dropped, it finds within it’s local neighbourhood whether an action node exists, if so connect to it’s input or output depending on which on is the closest. This means that users does not have to be absolutely accurate as to where to drop the circle on the action node. As long as the different of the distance between the circle and the action node is not above a certain amount, a connection could is established.

7.2.3.7 The drag and drop interface
The following is a screen shot of the whole process designer interface.

The large panel on the right hand side allows the user to construct the flow graph of action nodes by selecting individual action nodes and dropping the action node onto a location in the pane. We can see
in the panel a configuration that is defined, the function of this configuration is to search within a
database to see if there is an entry for the user name and password from the request and return the
returns in an XML document returned.

The drag and drop behaviour of the main panel is achieved through registering of individual
handlers of the nodes within the pane. These are defined in the common super class process nodes.
When a mouse dragged event is detected in a node, the node would then set the location of it self to the
new position of the mouse drag in the panel.

One the left hand side of the screen, two tables are used to display the input and output of the
selected node. This information can be obtained by registering a mouse click event on each of the
individual nodes, this event would then trigger an update of these tables.

The two buttons define step and delete node, is applicable to the currently selected node. So the
interface must also store the last node that the user has selected, so that the correct references could be
used to launch the editor for the node, or to delete the node.

### 7.2.3.8 Storage of Configuration Map using JDBC

The user can save definitions of in the process designer when completed constructing a particular
configuration. When the user presses the save configuration button on the designer, the designer first
checks the integrity and validity of the configuration, if no error is found, it will construct a database
statement to save the configuration in a binary format into the database. This involves:

1. Requesting every action node and connector within the flowchart to output an Element
   object representing details about that node, similar to the examples in section 6.5.
2. Then create a new Document object and add all the elements outputted by the action node
   and connectors to this document.
3. Then using the mobile application toolkit to output the document in binary XML format
   into a buffer.
4. Finally using a JDBC insert statement to insert the binary information contained in the
   buffer to the database as a binary BLOB.

In order to save the connection information into a XML document in step 1, it is first necessary to
assign a numerical identifier for every action node in the graph. Then each document that is
representative of a document must include the numerical identifier of the two nodes that the connector
is connected to.

### 7.2.4 Schema Designer

The database schema designer consists of a textual user interface that executes SQL statements. It’s
implementation is relatively straight forward. Basically whenever a user types in a statement and press
execute statement, the system executes the JDBC statement and returns the result back to the user. If an
SQL exception were raised, the exception is display to the user. A screen shot is shown below.
When a result is returned from the database, the designer checks whether the current statement expects a result. The way that the system knows this is through the type of SQL commands that the user has execute. If the statement is a SELECT statement, results would be displayed in a tabular form, similar to what exists in the screen shot. If on the other hand, a DELETE, INSERT or UPDATE statement is executed, the system would return a textual message of whether the query was successful or not.
8 Testing

The system was verified and validated using the following tests that I have designed to ensure that the specification is satisfied by its implementation. The functional test below is designed to test the system for defects.

8.1 Functional Testing

Functional testing also referred to as black-box testing is an approach to testing where the tests are derived from the program or component specification. In this case, our wireless system is seen as a black-box and all that we are concerned with is the system’s input test data and their corresponding output test results.

8.1.1 Elementary Testing of toolkit

A few basic tests have been carried out during the development of this toolkit. This involves creating a very basic application which sends an XML document in WBXML format, then creating another application that receives the WBXML format. This verifies that a large proportion of the basic system functions that I have specified in my specification is satisfied. This includes creating, editing and deletion of WBXML Documents, Elements and attributes. As well as sending and receiving the documents, which verifies that the encoder and parser are equivalent but performing the reverse process of each other.

When building the encoder and parser I have also verified that the byte code produced after encoding a document is correct, this is done by matching two XML document examples within the WBXML specifications with their encoded counterparts. I have made sure that my toolkit, given the same XML document, would produce the same byte code as those specified in the WAP specification. Additionally I have verified on other XML document structures that the byte code conform to the WBXML specification.

8.1.2 Functional Testing of toolkit using XML Server

A large part of the functionality testing of the mobile communication toolkit is already been carried out when I develop my XML server which uses this toolkit for both communication and configurations. The server performs all of the basic system functions that are identified in sections 3.1.1.1.1 to section 3.1.1.1.10.

8.1.3 Functional Testing using use case

One way to test the functionality of our system is to test the use cases identified in our specification. I have tested the scenario which is specified in section 3.3.2.2 by creating the configurations the represents searching of a stock quote given a symbol, and creating an application that sends a search request with a symbol specified. The following is the code for the application.
try {
    //open socket connection
    scrollText.addText("Connection Settings - socket://" + serverHost + ":1501
    thisSocket = (StreamConnection)Connector.open("socket://" + serverHost + ":1501", Connector.READ_WRITE);

    if (thisSocket == null) {  //check whether the connection is successfully established
        scrollText.addText("Couldn't open socket to server on port 1501...
        OpenServerSettingsDialog();
    } else {
        scrollText.addText("Connection is established to server.
    }

    // update the scrolltext display
    scrollText.paint();

    // get the input and output streams
    is = thisSocket.openInputStream();
    os = thisSocket.openOutputStream();
}

} catch (Exception e) {
    closeSocket();
    scrollText.addText("Server Not Avaliable. Please retry later.");
    scrollText.addText(e.toString());
    scrollText.paint();
}

The above fragment of code creates the socket connections. The follow fragment sends the stock_quote request document out to the XML server. The document has a root_element with tag stock_quote and one attribute with name Symbol and value SUN. Lines 1 to 5 creates and sends the request, lines 6 receives the reply and line 7 display the textual representation of the reply on the user interface.

1. WBXMLEncoder encoder = new WBXMLEncoder(os, g);
2. Element rootElement = new Element("stock_quote");
3. rootElement.addAttribute("SYMBOL", "SUN");
4. Document request = new Document(rootElement);
5. WBXMLOutputter.output(request, os);
6. Document reply = WBXMLBuilder.build(is);
7. scrollText.addText(reply.toString());

By modifying the contents of the database, different replies are received from the server, more over if there is not an entry of stock that has a symbol SUN, the server replies with an XML document with root element having a tag "No_Match".

This test scenario tests the following aspect of the server:

- Ability to perform database operations
• Ability to branch of define conditions
• Ability to send and receive XML documents
• Validity of the process designer
• Validity of the WBXML edit tools
• Functionality of schema editor
• Working database connectivity

It also test the following aspect of the mobile communication toolkit

• API to create and manipulate data structure to be sent
• Serialisation of data structure
• Communication between mobile device
• Can be used in a variety of mobile platforms
• Customisable functionality
• Process Tractability
• Contents identification

8.1.4 Functional Testing with mobile device
To ensure that the toolkit and server works with mobile device, I have tested their application on a Palm platform. A simple authentication configuration is first created on server using process designer. A screen shot is shown below.

The configuration defines an input document with a root element login_request, when this document is received by the server, the server would extract the attributes username and password from the document. Then the configuration defined a database operation to perform, namely to select from the table login, to see if there is an entry of this user name and password in the database. The result of this
operation is then passed to a conditional node which tests whether the results returned is an empty list or not. If so, proceed to the if branch, which sends out an XML document with root element login_reply and an attribute called authenticated having a value false. Meaning that authentication has failed. The else branch sends the same document out but with authenticated set to true.

Now it is necessary to create a mobile client application. The code for this could be found in the url \url{http://www.doc.ic.ac.uk/~fte97/project}. Essentially it consists of the following lines of code.

```
1. if(loginButton.pressed(x, y)) {
2.   try{
3.     WBXMLEncoder encoder = new WBXMLEncoder(os, g);
4.     Element rootElement = new Element("login_request");
5.     rootElement.addAttribute("username", tf.getText().trim());
6.     rootElement.addAttribute("password", tf2.getText());
7.     Document login = new Document(rootElement);
8.     WBXMLOutputter.output(login, os);
9.     scrollText.addText("Sent Encoded Message\n");
10.    scrollText.addText("Waiting for reply");
11.    Document reply = WBXMLBuilder.build(is);
12.    scrollText.addText(reply.toString());
13.  }
14. }catch(Exception e){
15.   e.printStackTrace();
16.  }
17. }
```

The body of the code is situated within an event handler, so line 1 of the code checks that the login button on the palm device is pressed. Lines 3 to 8 sends a login request out with user name and password set to the text entered into the two textboxes. Line 11 receives the reply and line 12 prints out the result.

Screen shots of the test application running is shown below.
We can see from the thread window of thread 1 that the server has relieved an XML document with root element of login_request and two attributes, they are the user name and password attributes, with values ‘ftc97’ and ‘pass’ respectively. The server then looks up its list of configurations for a match on the request, it has found 1 matching configuration. It then starts an SQL statement which returns a result being username=[] and password=[]., Which means that no results is found, this is correctly identified by the conditional and a reply is sent back to the mobile client, with authenticated = False in the attribute authenticated in the reply Document. Below is the mobile device’s view of this.

We can see the user name text box to be set to ftc97, and password is pass (with 4 asterisks). The scrollable message pane underneath the text boxes displays the reply that is obtained from the server. We can see here the string representation of the document with authenticated set to false.

Now to test the configuration, we try entering the right user name and password. The result on the palm device is shown below.
This time the correct user name and password is entered, the reply we obtained is an XML document with attribute authenticated set to true.

8.1.5 Interface Testing

Another test which I have conducted with 2 of my colleagues, is an interface test of the drag and drop interface in the process designer. I have provided them with a very basic brief of the functionality of the drag and drop server, and sat next to them as they try to create a sample configuration that I have described to them. The sample configuration is exactly the configuration defined in the previous section. Which authenticates a user given a user name and password. It was found that they constructed the configuration both within 15 minutes of use. When they were asked to insert the second configuration which is the stock quote search configuration, both managed to complete the configuration within 9 minutes of commerce. The feedback which I have obtained from them is that the drag and drop functionality is easy to understand, but it took them a bit more time to understand the input and output idea of each node and how these data are carried along with across with the flowchart. They also advised me to use connectors with only two circles instead of three, as three can make it confusing at times, to give the wrong idea to user that the middle circle would act as an end node.

8.2 Stress Testing

8.2.1 Stress Testing Routine

It is also necessary to test the saleability of the system. In our system, the speed of response of the central mobile information database is the most important factor to our system’s success, so it is necessary to carry out a few stress tests on our system. This can be done by building an application that simulates clients interacting with the XML server. The application can be built by using some code from the mobile client application, and looping through to send and receive an XML document. Here is a high level pseudo-code to our testing application.
Take as input the number of clients to simulate, say $N$ and how many times each client needs to access the server — $M$

Create $N$ number of threads
In each thread run the following:
   Login client
   Loop $M$ times
      Create An WBXML Element
      Send the WBXML ELEMNT
      RECEIVE WBXML REPLY
      Record respond time
   End loop

The concrete implementation of this stress testing class could be found in appendix 7. The code essentially creates $N$ number of threads, and starts all the threads running together. Within each thread body, the thread creates a document representing a request. In this case we create a login_request document. The thread then records the current time in milliseconds. The thread then sends this document to the server and waits for a reply. Once a reply has been received, the thread records again the time. The difference between the first and the second time is the time the thread to send this request + time it takes for server to receive this document, + time it takes for server to handle operations defined by this document + time it takes for the reply be sent by the server and received by the thread.

8.2.2 Testing Environment

The test is run on two separate machines. Both the StressTest client and the XML server is run on an Intel P2 400 megahertz PC with 128 MB RAM. Both machines uses the Windows 2000 operating system and is running on Java Virtual Machine version 1.2.2. The two PC has no applications actively opened and has both a minimal amount of background tasks running. The two PC’s are connected using a LAN with transmission rate of 100 mega bits per seconds.
8.2.3 Stress test results

The following table shows the result of obtaining from running the stress test code in appendix 7 on the server under the environment defined.

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>Average of time taken for all threads</th>
<th>Average of time taken per thread</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1012</td>
<td>1012</td>
</tr>
<tr>
<td>2</td>
<td>4666</td>
<td>2333</td>
</tr>
<tr>
<td>3</td>
<td>5019</td>
<td>1673</td>
</tr>
<tr>
<td>4</td>
<td>10754</td>
<td>2688.5</td>
</tr>
<tr>
<td>5</td>
<td>14372</td>
<td>2874.4</td>
</tr>
<tr>
<td>6</td>
<td>17537</td>
<td>2922.83333</td>
</tr>
<tr>
<td>7</td>
<td>23512</td>
<td>3358.857143</td>
</tr>
<tr>
<td>8</td>
<td>32066</td>
<td>4008.25</td>
</tr>
<tr>
<td>9</td>
<td>32307</td>
<td>3589.666667</td>
</tr>
<tr>
<td>10</td>
<td>36565</td>
<td>3656.5</td>
</tr>
<tr>
<td>11</td>
<td>46709</td>
<td>4246.272727</td>
</tr>
<tr>
<td>12</td>
<td>55501</td>
<td>4625.08333</td>
</tr>
<tr>
<td>13</td>
<td>61160</td>
<td>4704.615385</td>
</tr>
<tr>
<td>14</td>
<td>68193</td>
<td>4870.928571</td>
</tr>
<tr>
<td>15</td>
<td>72012</td>
<td>4800.8</td>
</tr>
</tbody>
</table>

The test is run 5 times, each time the total time that is required for each thread to complete the send and receive operation is added together. The second column shows the average of the time obtained from running the test 5 times. The last column is calculated by dividing the second column by the first, i.e., dividing the total amount of time required for all threads by the number of threads there is. This gives us the average time for one thread to complete the send and receive operation. The following graph plots this result more clearly.
We can see from the graph that there is a clear trend that as we increase the number of simultaneous threads accessing the server, the average time of required to process this thread increases. We can see from the graph that if 1 thread is accessing the server, the server takes around 1 second to respond. This is very good and well within the 5 second threshold that we have specified for reaction time in our specification. However as we increase the number of simultaneous threads accessing the server, we see average processing time to gradually increase. Until we have reached nearly 5 seconds for 15 simultaneous threads. This is approaching our 5 seconds threads hold value.

It should be noted that although the results from this simulation shows that around 15 threads is the acceptance number of active connections that the server can establish, in actual fact, our server should be able to handle much more clients than that. This is because in this process here, all 15 threads access the server at the same time, for the same resource. So inevitably, the server would have to create 15 handler threads and perform all database queries simultaneously, this would take up much more resources than it’s usual access pattern. In reality, there should be a lot of active client connections, but not more than a couple clients would be requesting for an operation at any one time. But of course the pattern of access to the server is very application dependent. Similarly the type of configurations that’s define is very application dependent, and different type of configurations should take different time to process. Never the less, this stress testing method provide us with a good idea of how well our server is at cooping with heavy access traffic.

8.2.4 Possible sources of bias

This stress testing method is a very rudimentary one, there are many possible sources of bias which we have not taken into account. The result of the test should only provide us with a rough idea of our server’s performance. Possible sources of bias, includes the assumption that all clients would be performing a select operation in the database, the assumption that the time it takes to create a new data object is negligible, “assumption that our operating system and java virtual machine arrange for similar amount of resources for every test that we carried out” and the assumption that the performance of our network stack does not limit the performance of send and receive operations of the thread. In reality most of these matters. These sources of bias may have contributed to the slight fluctuation in results of the graph, as sometimes average time decreases as we have more threads requesting.

8.3 WBXML Protocol Efficiency

I have also gathered some rough statistics as to how much efficiency the WBXML protocol provides us over the normal text based document transmission. For the two XML documents and their WBXML equivalent that is shown in appendix 8 and 9. In the first document, the plain text form has 488 bytes whilst the WBXML equivalent has only 57 bytes, this is only 12 % the size of the original document. The the second document, which has a smaller size, the original has 171 bytes and it’s WBXML form has only 33 bytes. This time the size is 20% of the original. In general the larger the document size, the greater the efficiency this is because of the use of the string table to allow repeated strings to be referenced from the document. The efficiency gain depends also on the actual application and structure, but it has been said that on average the size of the WBXML document is about 20% to 25% the size of the original. This means more than four times faster transmission of XML documents of wireless devices.
9 Evaluation

This chapter considers the key achievements and limitation of this toolkit and XML server system. It also would discuss the current and future applications of the system as well as a conclusion of the insights that is gained from this project.

9.1 Achievements

In this section I assess what I have achieved in this project and to what extent have I achieved what I originally wanted to build. The following is an evaluation of how well I have satisfied the system functions that I have identified in my specification in section 3 for my toolkit and XML server.

9.1.1 Mobile Communication Toolkit

<table>
<thead>
<tr>
<th>System Function</th>
<th>How well I have met the requirement?</th>
</tr>
</thead>
<tbody>
<tr>
<td>API to create and manipulate data structure to be sent</td>
<td>I have created an API based of DOM (Document Object Model) to construct and manipulate an XML document. Some of the DOM functionality has to be comprised in the design phase due to the limited support of the WAP binary XML protocol for meta-level information. Never the less, I have designed a fully functional API to manipulate XML entities, including elements, attributes and name space.</td>
</tr>
<tr>
<td>Good authoring tool for content management</td>
<td>The ability of outputting documents into a binary format and storing it to a database facilitates its use in content management. The tool provides a simple API to manipulate XML documents, however in some applications it may be easier to use just a plain document that does not need the hierarchical structure. It can be argued that in this case, XML does not provide a good authoring tool.</td>
</tr>
<tr>
<td>Serialisation of data structure</td>
<td>Implementation of the WBXML encoder has allow simple and systematic tool to serialise an XML document.</td>
</tr>
<tr>
<td>Communication between mobile device</td>
<td>The use of binary XML format to communicate between devices has been well tested here, specially in its application on the XML server. My testing procedures highlighted in chapter 8 has made sure that the implementation is correct and works on mobile devices.</td>
</tr>
<tr>
<td>Can be used in a variety of mobile platforms</td>
<td>The use of J2ME technology to develop our toolkit has allowed us to achieve this.</td>
</tr>
<tr>
<td>Can be used in a variety of network infrastructure</td>
<td>Again J2ME has abstracted away this problem from our toolkit. The kilobyte virtual machine which runs on different devices provides us with a standard interface to send and receive bits from the network.</td>
</tr>
<tr>
<td>Easy to debug</td>
<td>The implementation of KJDOMException has allowed application programmers to catch run time error during use of my toolkit. However the problem with debugging on mobile devices is that there is not a console to print error messages for debug, therefore the use of my approach is only useful if somehow the application developer can read</td>
</tr>
</tbody>
</table>
back the error messages that has been raised.

Customisable functionality

I have used the visitor pattern to achieve a clear separation between document manipulation and creation with document encoding and parsing. An interface called parser and encoder is provided to allow custom parsing and encoding of a document.

Contents identification

Implemented a hash function that returns an integer value depending on the content of an XML document. But the function differentiates between documents with elements that is arranged in different order, this can be a problem for developers.

Compression

Although a frame work has been provided to allow compression streams, a concrete implementation has not been implemented. An area not investigated is how viable compression is for the limited processing capability of mobile devices.

Asynchronous Receive

My toolkit does not directly provide this function due to the limitations of threads in the KVM.

End to End Encryption

Like compression, a concrete implementation has not been included.

Pluggable layers of communication

This is fully satisfied using the abstract classes encode and parser.

9.1.2 XML Server

<table>
<thead>
<tr>
<th>System Function</th>
<th>How well I have met the requirement?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ability to send and receive data structure from mobile device</td>
<td>This is automatically achieved with the completion of my communication toolkit.</td>
</tr>
<tr>
<td>Definition of data structure that server is allowed to accept</td>
<td>This functionality is full implemented.</td>
</tr>
<tr>
<td>Manipulation of database schema</td>
<td>A basic implementation has been provided since alternative database managers could be used from database vendors.</td>
</tr>
<tr>
<td>Template Definition using a “drag and drop” interface</td>
<td>Fully Implemented using framework provided by Java Swing.</td>
</tr>
<tr>
<td>Long term storage of server configurations</td>
<td>Achieved using binary communication toolkit and JDBC.</td>
</tr>
<tr>
<td>Handling of data structure request</td>
<td>Fully multithread server implemented.</td>
</tr>
<tr>
<td>Allow encoding of conditionals and loops in the template</td>
<td>Conditionals has been implemented but loops has not. Although the frame work is there to implement a loop action node similar to the four that is already in place.</td>
</tr>
<tr>
<td>Dynamic Configuration</td>
<td>Implemented, configurations via database.</td>
</tr>
<tr>
<td>Traffic and access statistics</td>
<td>Basic functionality implemented to log accesses to files. However, high level forms of feedback such as graphs and summary has not been included in server. A separate software could be built to analyse the log.</td>
</tr>
</tbody>
</table>
9.2 What it can do
9.2.1 Key Benefits

My toolkit provides an **application and platform independent** tool for encoding and parsing XML in a **compact** format using mobile devices. It allows **simple and fast manipulation** of XML documents with **error tractability**.

The XML server is a **multi-threaded** application that provides a **drag and drop interface** for manipulations of request handling procedures. The server has **database capability** and is associated with a set of **dynamically defined configurations** which defines the behaviour of the server.

9.2.2 Applications

The toolkit could be applied in scenarios of mobile application development that requires communications capabilities. Mobile applications developers can save time from programming byte level communication streams into their applications. It can also be used to easily add XML parsing and encoding capability to mobile applications. It’s real world application can range anything from performing the communications for a customisable search engine to an ‘intelligent’ micro-wave talking to a refrigeration. It is applicable in all devices that has a Kilobyte Virtual Machine built into it, regardless of the platform, micro processor make, kind of wireless networks that is in use. The future of this technology is exciting and broad, already mobile phones are being sold now which comes with a built in KVM that conforms to the J2ME standard. This means that they can install my toolkit (which has a size of about 200 kilobytes) into their device and communicate in WAP XML format with another device. The fact that the toolkit is not tied to any particular applications, gives it portability in it’s use in a very wide variety of fields.

The server could be used in applications that requires highly customised behaviours. In situations where requests has to be handled individually, or in situations where the logic of the application is continually changing. For example, our server could be used in the field of finance to constantly monitor market activities of stock performances. When the price of the stock is above or below a certain threshold, the server would notify a user via a WBXML document. We can easily create a configuration to perform this task using the drag and drop interface of the process designer. If later on, we want to change the actual values of these threshold or if we want to notify more users or we may even want the server to automatically buy and sell these shares for us by sending off buy and sell requests to a different location, we can do all that without writing a single line of code and without recompiling any classes within our server. This is the **key** advantage of our server. That all the configurations for the handling process is defined at **run time** using a graphical interface. It’s application can mainly be for interfacing the database operations to mobile clients but the **framework** that this server is built on can be used in a very wide variety of applications.
9.3 Limitations

I feel that the main limitation of my toolkit is the fact that it is tied to the J2ME technology. The successful of the use of this toolkit in a wide variety of mobile devices depends heavily on the pre condition that J2ME is going to take off the way it promises within the next few years. J2ME is still a very pre-mature technology that is gaining in momentum, so the standard is bound to take on another phase of major change. It may even be the fact that mobile technology develops up to a stage where the standard Java Virtual machine can run on any API in which case J2ME would become redundant. Although I feel that this is a weakness of my toolkit, I think that I have made the correct decision in choosing J2ME as my development language, this is because no other environment provided me similar platform independency that I wanted.

Another limitation of my toolkit is it’s lack of support for meta level information, the reason being that it is not supported by the WBXML standard. I feel that it would be easier if I am an application developer and able to define constraints for XML documents that I send off and receive. This raises the other question, was WBXML the right choice of protocol standard to use? I feel that it was, because it was the only standard that I could find at that time that would give me the compact form of encoding that current mobile technologies desperately needs. With the speed of development of wireless technology, I have no doubt that this eventually will not be the primary concern, but at the moment with most wireless connections only 9.6kbps, sending and receiving full XML documents would cause too much redundancy to network efficiency.

In terms of my XML server, I feel that it’s main limitation is lack of support for loops which I did not have enough time to complete implementing. Without loops, it is difficult to define constructs that is reoccurring, it makes it also very difficult to handle lists of information which is received from XML documents or that is extracted from the database. The proper infrastructure however is there to implement this feature given more time.

Another limitation of the server that I have not addressed is the inability to perform database operations from multiple databases, which would be very useful in situations of system integrations. It is not unusual for information to be pulled from several database before it is brought together for transmission.

9.4 Conclusion

The major challenge in this project is the fact that I am trying to build in a field mobile application development that is undergoing a very fast period of change. The requirements and assumptions made is also changing with the field. For example, the assumption that I made during the start that bandwidth limitations and processor limitations was the primary concern at the time, but as more and more powerful devices are released and speed of wireless connectivity increases, these are no longer the primary concerns. Therefore design decisions might also change. For example in choosing an XML communication protocol to implement, a trade off has to be taken between compactness and functionalities. These design decisions can vary as our priorities and the situation changes. This is the main thing that I learnt in the project. And the longer I work in this project the more I realise the importance of a general framework is to the development of any applications. The reasons for example for me being able to implement the connector class so
elegantly in my process designer by inheriting a JComponent class, is due to the framework that was established for Swing.

Apart from the above insights, I feel that also there are other insights that could be gained from this project in the field of mobile application development. The importance of platform independence in the present world of heterogeneous mobile device, as well as the power of dynamically configurable server in constantly changing environment. My toolkit can save mobile application developer’s time on building low level communication details. My server can offer a flexible way of integrating mobile applications data, collaborating information together, collected from heterogeneous devices into a single point of storage and access.

Despite the fast changing field of mobile application development, this project has never the less this project has provided insights to architectures which might be possible for work related to collaborating mobile information together. The traditional distinction between mobile devices and personal computers are also becoming blurred as can be seen in the application of my toolkit where a single piece of code is run on a mobile devices then on a PC without changing a single line of code. Similarly, I hope the concept of my XML server, and it’s ability to dynamically change behaviour has provided an insight to what can be possible in the field of mobile application development.

9.5 Further Development Possibilities

This last section describes how this project can be taken further given more time and resources.

9.5.1 Inclusion of Loops

One of the limitations that I have mentioned in my server is it’s inability to encode loops as one of it’s elementary actions. The inclusion of loops allows developers to capture reoccurring operations within the configuration definitions. To include loop, a new class has to be created that extends the class process nodes, this will inherit all the basic drag and drop functionalities that comes with process nodes. As well as definitions of input and output data. The remaining development lies on designing a user interface to define the loop and open this dialog when the openEditPanel() method is called, which is a method that has to be overidded from process nodes. The last step is to change the XML server so that it recognises loop as an action node.

9.5.2 Use of a standard text based XML protocol

It would be ideal if application developers using the toolkit can choose between the more compact WBXML protocol or another XML protocol such as SOAP which uses text based encoding. This allows more interoperability of the server and toolkit with other legacy systems.

9.5.3 Inclusion of standard encryption and compression implementation

It would also enhance the value of the toolkit if standard compression and encryption algorithms are implemented to the toolkit.
9.5.4 Inclusion of Meta Information

Given that the protocol supports the transmission of meta level instructions for an XML document, we can include support for DTD and XML schema, so that application developers can apply constraint to XML the document that is being manipulated.

9.5.5 Implementation of extended features in DOM level 1, 2

Given more time extended features in DOM level 1 and 2 would be implemented to provide a more extensive API for XML manipulation.

9.5.6 Adapter with standard XML parsers

Due to the similarity of structure and API, we can actually create an adapter that translates our XML document objects into a document object of a standard parser. For example the Apache Xerces XML parser. If this can be done, we can use easily transform our XML documents to different forms using the XML transformer functions of the package.

9.5.7 XMLT engine for mobile device

We can possibly build a XMLT engine which transforms our XML document into another format, for example WML or HTML, given a style sheet. This allows more extensive applications of our toolkit, as well as providing more ways for clients to access our server.

9.5.8 Custom code node

An idea that I have thought about is to allow users to define custom code within an action node in the server. So a user interface allows user to type in their code in a function, the input and outputs to the nodes are parameters to the function. User can then manipulate the input and output using Java code which is loaded and verified at run time. This way we provide users with the power of the whole language to use.

9.5.9 Developing into a commercial product

The server and the toolkit can be developed into a commercial middleware package. With the wireless field expanding so quickly at the present, I think that there is a market for mobile application middleware. This is because firstly there is still very little tool available, secondly, in this field we need a clear separation between what is commonly referred to as “business logic” with the actual applications. Business logic are the configurations which my server loads up instead of having it built in, which is what most products in the market currently does.
### Bibliography

<table>
<thead>
<tr>
<th>Title</th>
<th>Author/Site</th>
<th>Publisher</th>
<th>Issue No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Applying UML and Patterns</td>
<td>Criag Larman</td>
<td>Prentice Hall</td>
<td>1</td>
</tr>
<tr>
<td>Design Patterns</td>
<td>Erich Gamme</td>
<td>Addison Wesley</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Richard Helm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JAVA and XML</td>
<td>Brett McLaughlin</td>
<td>O'Reilly</td>
<td>1</td>
</tr>
<tr>
<td>Software engineering : a practitioner's approach</td>
<td>Roger S. Pressman</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Requirements engineering and rapid development : an object-oriented approach</td>
<td>Ian Graham</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>OPEN modeling with UML</td>
<td>Brian Henderson-Sellers and Bhuvan Unhelkar</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Applied cryptography : protocols, algorithms and source code in C</td>
<td>Bruce Schneier</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Non-functional requirements in software engineering</td>
<td>Lawrence Chung</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Professional WAP</td>
<td>Charles Arehart</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>XML : principles, tools, and techniques</td>
<td>Dan Connolly</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Inside WAP : programming applications with WML and WMLScript</td>
<td>Pekka Niskanen</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Java Threads</td>
<td>Scott Oak</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Java Security</td>
<td>Scott Oaks</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Java Network Programming</td>
<td>Merlin Hughes</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td><a href="http://www.w3.org/TR/wbxml">http://www.w3.org/TR/wbxml</a></td>
<td>WWW Consortium</td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="http://www.java.sun.com/">http://www.java.sun.com/</a></td>
<td>Sun micro system’s website</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
http://www.mertowerks.com: Code warrior’s web site

http://www.palm.com: Palm Computing Website

http://www.microsoft.com: Microsoft

http://alphaworks.ibm.com/tech/xml4j: IBM ALPHAWORKS website

http://www.w3.org/TR/SOAP: SOAP

http://www.wddx.org/: WDDX Protocol


http://www.embedded.oti.com/: About Embedded devices

http://www.lotus.com.: Lotus notes website

http://www.apache.com: Apache Xerec Parser
11 Appendix

11.1 Appendix 1

Below is the explanation of the features of the definitions in the columns above.

**Status** – The stage that the XML standard is at in development.

**XML Schema** – Whether the standard supports using XML schema as a means of constraining the document in its present applications.

**DTD** – Whether Data type Definitions are currently used to constrain the document type.

**Fixed Serialisation** – Generated XML that uses fixed element names to serialize application data structures.

**Automatic Serialisation** – Generated XML that uses element names derived from application data structures through some type of mapping or rules.

**Implicit Data typing** - does not carry detailed data types in the serialization or an external schema.

**Explicit Data typing** - data types with elements or in an external schema.

**Extensibility** – Protocol provides a mechanism that allows extending of the functionality of the XML document that is application independent.

**Compression** – Defines the mode that the XML document is encrypted in.

**RPC** - Protocol provides means for remote procedure calling.
### 11.2 Appendix 2

The following shows the document structure of WBXML in its BNF form, source obtained from [http://www.w3.org/wbxml](http://www.w3.org/wbxml)

A binary XML document is composed of a sequence of elements. Each element may have zero or more attributes and may contain embedded content.

```
start = version publicid charset strtbl body
strtbl = length *byte
body = *pi element *pi
element = stag [ 1*attribute END ] [ *content END ]
content = element | string | extension | entity | pi | opaque
stag = TAG | ( LITERAL index )
attribute = attrStart *attrValue
attrStart = ATTRSTART | ( LITERAL index )
attrValue = ATTRVALUE | string | extension | entity
extension = ( EXT_I termstr ) | ( EXT_T index ) | EXT
string = inline | tableref
inline = STR_I termstr
tableref = STR_T index
entity = ENTITY entcode
entcode = mb_u_int32 // UCS-4 character code
pi = PI attrStart *attrValue END
opaque = OPAQUE length *byte
version = u_int8 containing WBXML version number
publicid = mb_u_int32 | ( zero index )
charset = mb_u_int32
termstr = charset-dependent string with termination
index = mb_u_int32 // integer index into string table.
length = mb_u_int32 // integer length.
zero = u_int8 // containing the value zero
(0).
```
11.3 Appendix 3

The following details of encoding application tokens are found from www consortium website.

11.3.1 Tag Code Space

Tag tokens are a single u_int8 and are structured as follows:

Table 3. Tag format

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 (most significant)</td>
<td>Indicates whether attributes follow the tag code. If this bit is zero, the tag contains no attributes. If this bit is one, the tag is followed immediately by one or more attributes. The attribute list is terminated by an END token.</td>
</tr>
<tr>
<td>6</td>
<td>Indicates whether this tag begins an element containing content. If this bit is zero, the tag contains no content and no end tag. If this bit is one, the tag is followed by any content it contains and is terminated by an END token.</td>
</tr>
<tr>
<td>5 - 0</td>
<td>Indicates the tag identity.</td>
</tr>
</tbody>
</table>

For example:

- Tag value 0xC6: indicates tag six (6), with both attributes and content following the tag, eg, `<TAG arg="1">foo</TAG>`.  
- Tag value 0x46: indicates tag six (6), with content following the start tag. This element contains no attributes, eg. `<TAG>test</TAG>`.  
- Tag value 0x06: indicates tag six (6). This element contains no content and has no attributes, eg, `<TAG/>`.  

The globally unique code LITERAL (see section 5.8.4.5) represents unknown tag names. An XML tokeniser should avoid the use of the LITERAL or string representations of a tag when a more compact form is available.

Tags containing both attributes and content always encode the attributes before the content.

11.3.1.2 Attribute Code Space (ATTRSTART and ATTRVALUE)

Attribute tokens are encoded as a single u_int8. The attribute code space is split into two ranges (in addition to the global range present in all code spaces):

- Attribute Start - tokens with a value less than 128 indicate the start of an attribute. The attribute start token fully identifies the attribute name, eg. URL=, and may optionally specify the beginning of the attribute value, eg. PUBLIC="TRUE". Unknown attribute names are encoded with the globally unique code LITERAL (see section 5.8.4.4). LITERAL must not be used to encode any portion of an attribute value.  
- Attribute Value - tokens with a value of 128 or greater represent a well-known string present in an attribute value. These tokens may only be used to represent attribute values. Unknown attribute values are encoded with string, entity or extension codes (see section 5.8.4).

All tokenised attributes must begin with a single attribute start token and may be followed by zero or more attribute value, string, entity or extension tokens. An attribute start token, a LITERAL token or the END token indicates the end of an attribute value. This allows a compact encoding of strings containing well-known sub-strings and entities.

For example, if the attribute start token TOKEN_URL represents the attribute name "URL", the attribute value token TOKEN_COM represents the string ".com" and the attribute value token TOKEN_HTTP represents the string "http://", the attribute URL="http://foo.com/x" might be encoded with the following sequence:

TOKEN_URL TOKEN_HTTP STR_I "foo" TOKEN_COM STR_I "/x"

In another example, if the attribute start token TOKEN_PUBLIC_TRUE represents the attribute name "PUBLIC" and the value prefix "TRUE", the attribute PUBLIC="TRUE" might be encoded with the following sequence:

TOKEN_PUBLIC_TRUE

An XML tokeniser should avoid the use of the LITERAL or string representations of an attribute name when a more compact form is available. An XML tokeniser should avoid the use of string representations of a value when a more compact form is available.
11.4 Appendix 4

The following details of encoding global tokens are found from www consortium website.

11.4.1.1 Strings

string = inline | tableref
inline = STR_I termstr
tableref = STR_T index

Strings encode inline character data or references into a string table. The string table is a concatenation of individual strings. String termination is dependent on the character document encoding and should not be presumed to include NULL termination. References to each string include an offset into the table, indicating the string being referenced.

Inline string references have the following format:

```
STR_I termstr
```

String table references have the following format:

```
STR_T index
```

The string table offset is from the first byte of the first string in the table (ie, not a character offset).

11.4.1.2 Global Extension Tokens

extension = ( EXT_I termstr ) | ( EXT_T index ) | EXT

The global extension tokens are available for document-specific use. The semantics of the tokens are defined only within the context of a particular document type, but the format is well defined across all documents. There are three classes of global extension tokens: single-byte extension tokens, inline string extension tokens and inline integer extension tokens.

Inline string extension tokens (EXT_I*) have the following format:

```
EXT_I* termstr
```

Inline integer extension tokens (EXT_T*) have the following format:

```
EXT_T* index
```

Single-byte extension tokens (EXT*) have the following format:

```
EXT*
```

11.4.1.3 Character Entity

entity = ENTITY entcode
entcode = mb_u_int32 // UCS-4 character code

The character entity token (ENTITY) encodes a numeric character entity. This has the same semantics as an XML numeric character entity (eg, &amp;#32). The mb_u_int32 refers to a character in the UCS-4 character encoding. All entities in the source XML document must be represented using either a string token (eg, STR_I or the ENTITY token.

The format of the character entity is:
11.4.1.1.4 Processing Instruction

pi = PI attrStart *attrValue END

The processing instruction (PI) token encodes an XML processing instruction. The encoded PI has identical semantics to an XML PI. The attrStart encodes the PITarget and the attrValue encodes the PI's optional value. For more details on processing instructions, see [XML].

The format of the PI tag is:

\[
\text{PI} \quad \text{attrStart} \quad \text{attrValue} \quad \text{END}
\]

PIs without a value are encoded as:

\[
\text{PI} \quad \text{attrStart} \quad \text{END}
\]

11.4.1.1.5 Literal Tag or Attribute Name

The literal token encodes a tag or attribute name that does not have a well-known token code. The actual meaning of the token (ie, tag versus attribute name) is determined by the token parsing state. All literal tokens indicate a reference into the string table, which contains the actual name.

The format of the LITERAL tag is:

\[
\text{LITERAL} \quad \text{mb_u_int32}
\]

11.4.1.1.6 Opaque Data

opaque = OPAQUE length *byte

The opaque token (OPAQUE) encodes application-specific data. A length field and zero or more bytes of data follow the token. The length field encodes the number of bytes of data, excluding the OPAQUE token and the length field.

11.4.1.1.7 Miscellaneous Control Codes

11.4.1.1.7.1 END Token

The END token is used to terminate attribute lists and elements. END is a single-byte token.

11.4.1.1.7.2 Code Page Switch Token

The code-page switch token (SWITCH_PAGE) indicates a switch in the current code page for the current token state. The code-page switch is encoded as a two-byte sequence:

\[
\text{SWITCH} \quad \text{u_int8}
\]

11.4.1.1.8 Reserved Tokens

There are several reserved global tokens. These must not be emitted by a tokeniser and should be treated as a single-byte token by a user agent.

The code page 255 is reserved for implementation-specific or experimental use. The tokens in this code page will never be used to represent standard XML document constructs.
11.5 Appendix 5

The following is an example of encoding an XML document into its WBXML equivalent.

Source XML Document:

```xml
<?xml version="1.0"?>
<!DOCTYPE XYZ [
<!ELEMENT XYZ ( CARD )+ >
<!ELEMENT CARD (#PCDATA | INPUT | DO)*>
<!ATTLIST CARD NAME NMTOKEN #IMPLIED>
<!ATTLIST CARD STYLE (LIST|SET) 'LIST'>
<!ATTLIST DO TYPE CDATA #REQUIRED>
<!ATTLIST DO URL CDATA #IMPLIED>
<!ELEMENT INPUT EMPTY>
<!ATTLIST INPUT TYPE (TEXT|PASSWORD)'TEXT'>
<!ATTLIST INPUT KEY NMTOKEN #IMPLIED>
<!ENTITY nbsp "&#160;">]
<!-- This is a comment -->
<XYZ>
  <CARD NAME="abc" STYLE="LIST">
    <DO TYPE="ACCEPT" URL="http://xyz.org/s"/>
    Enter name: <INPUT TYPE="TEXT" KEY="N"/>
  </CARD>
</XYZ>
```

The following tokens are defined for the tag code space:

<table>
<thead>
<tr>
<th>Tag Name</th>
<th>Token</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARD</td>
<td>5</td>
</tr>
<tr>
<td>INPUT</td>
<td>6</td>
</tr>
<tr>
<td>XYZ</td>
<td>7</td>
</tr>
<tr>
<td>DO</td>
<td>8</td>
</tr>
</tbody>
</table>

The following attribute start tokens are defined:

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Value Prefix</th>
<th>Token</th>
</tr>
</thead>
<tbody>
<tr>
<td>STYLE</td>
<td>LIST</td>
<td>5</td>
</tr>
<tr>
<td>TYPE</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>TYPE</td>
<td>TEXT</td>
<td>7</td>
</tr>
<tr>
<td>URL</td>
<td>http://</td>
<td>8</td>
</tr>
<tr>
<td>NAME</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>KEY</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>

The following attribute value tokens are defined:

<table>
<thead>
<tr>
<th>Attribute Value</th>
<th>Token</th>
</tr>
</thead>
<tbody>
<tr>
<td>.org</td>
<td>85</td>
</tr>
<tr>
<td>ACCEPT</td>
<td>86</td>
</tr>
</tbody>
</table>

Tokenised form (numbers in hexadecimal) follows. This example assumes an UTF-8 character encoding and NULL terminated strings:

```
01 01 6A 12 'a' 'b' 'c' '0' '0' '0' 'E' 'n' 't' 'e' 'r' 'n' 'a' 'm' 'e' ':' ' ' 00 47 C5 09 03 00 05 01 88 06 86 08 03 'x' 'y' 'z' '0' '0' '8' '5' '0' '3' '7' 'x' '0' '0' '1' '8' '3' '0' 01 83 04 86 07 0A 03 'N' '0' '0' '0' '1' '0' '1' 01
```

In an expanded and annotated form:
<table>
<thead>
<tr>
<th>Token Stream</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Version number - WBXML version 1.1</td>
</tr>
<tr>
<td>01</td>
<td>Unknown public identifier</td>
</tr>
<tr>
<td>6A</td>
<td>charset=UTF-8 (MIBEnum is 106)</td>
</tr>
<tr>
<td>12</td>
<td>String table length</td>
</tr>
<tr>
<td>a', b', c', 00, '&quot;', 'E', 'n', 't', 'e', 'r', ' ', 'n', 'a', 'm', 'e', ':', ' ', 00</td>
<td>String table</td>
</tr>
<tr>
<td>47</td>
<td>XYZ, with content</td>
</tr>
<tr>
<td>C5</td>
<td>CARD, with content and attributes</td>
</tr>
<tr>
<td>09</td>
<td>NAME=</td>
</tr>
<tr>
<td>83</td>
<td>String table reference follows</td>
</tr>
<tr>
<td>00</td>
<td>String table index</td>
</tr>
<tr>
<td>05</td>
<td>STYLE=&quot;LIST&quot;</td>
</tr>
<tr>
<td>01</td>
<td>END (of CARD attribute list)</td>
</tr>
<tr>
<td>88</td>
<td>DO, with attributes</td>
</tr>
<tr>
<td>06</td>
<td>TYPE=</td>
</tr>
<tr>
<td>86</td>
<td>ACCEPT</td>
</tr>
<tr>
<td>08</td>
<td>URL=&quot;http://&quot;</td>
</tr>
<tr>
<td>03</td>
<td>Inline string follows</td>
</tr>
<tr>
<td>x', y', z', 00</td>
<td>string</td>
</tr>
<tr>
<td>85</td>
<td>&quot;.org&quot;</td>
</tr>
<tr>
<td>03</td>
<td>Inline string follows</td>
</tr>
<tr>
<td>?, ', 00</td>
<td>string</td>
</tr>
<tr>
<td>01</td>
<td>END (of DO attribute list)</td>
</tr>
<tr>
<td>83</td>
<td>String table reference follows</td>
</tr>
<tr>
<td>04</td>
<td>String table index</td>
</tr>
<tr>
<td>86</td>
<td>INPUT, with attributes</td>
</tr>
<tr>
<td>07</td>
<td>TYPE=&quot;TEXT&quot;</td>
</tr>
<tr>
<td>0A</td>
<td>KEY=</td>
</tr>
<tr>
<td>03</td>
<td>Inline string follows</td>
</tr>
<tr>
<td>N', 00</td>
<td>String</td>
</tr>
<tr>
<td>01</td>
<td>END (of INPUT attribute list)</td>
</tr>
<tr>
<td>01</td>
<td>END (of CARD element)</td>
</tr>
<tr>
<td>01</td>
<td>END (of XYZ element)</td>
</tr>
</tbody>
</table>
11.6 Appendix 6

The following table shows the characteristics and technology of different wireless networks. Source of information if found in a enterprise white paper at Microsoft’s website.

<table>
<thead>
<tr>
<th>Wide Area Air Links</th>
<th>Carrier/Company Examples</th>
<th>Actual Bandwidth</th>
<th>Coverage</th>
<th>Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell phone</td>
<td>VoiceStream Wireless</td>
<td>Up to 28.8 Kbps</td>
<td>5,500 U.S. and Canadian cities</td>
<td>U.S.-based GSM and GPRS</td>
</tr>
<tr>
<td>Cell phone</td>
<td>SprintPCS</td>
<td>14.4 Kbps</td>
<td>300 major metropolitan areas, 4,000 U.S. cities</td>
<td>CDMA</td>
</tr>
<tr>
<td>Cell phone</td>
<td>British Telecom</td>
<td>Up to 28.8 Kbps</td>
<td>Trial phase, not yet commercially available</td>
<td>European-based GSM and GPRS</td>
</tr>
<tr>
<td>Data network</td>
<td>Motient (American Mobile Satellite Corp. and Ardis), Mobitex</td>
<td>9.6 Kbps</td>
<td>99 percent of the 500 most populated U.S. cities (220 million people)</td>
<td>Two-way wireless packet data</td>
</tr>
<tr>
<td>High-bandwidth data</td>
<td>Metricom</td>
<td>128+ Kbps</td>
<td>45 metropolitan areas (100 million people) by the end of 2001</td>
<td>Ricochet MicroCellular 128K Network</td>
</tr>
<tr>
<td>Paging data</td>
<td>AT&amp;T Wireless, Verizon Communications (Bell Atlantic Mobile, GTE Wireless, a portion of Vodafone AirTouch)</td>
<td>9.6 Kbps</td>
<td>90 percent of the U.S. population</td>
<td>CDPD, Reflex</td>
</tr>
<tr>
<td>Satellite data</td>
<td>Norcom Networks</td>
<td>Up to 2 Kbps</td>
<td>Trial phase in early 2001, commercially available in North America H2 2001</td>
<td>Integrated 802.11b, terrestrial and satellite network access</td>
</tr>
<tr>
<td>Reseller, systems integrator</td>
<td>GoAmerica, OmniSky, UUNET UUMobile, WirelessWebConnect, WorldCom Wireless Internet</td>
<td>9.6 Kbps ?128+ Kbps</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
import com.sun.kjava.*;
import java.util.Vector;
import java.io.*;
import org.kjdom.*;
import java.net.*;
import java.util.*;
import javax.swing.SwingUtilities;

public class StressTest{
    long totalTime;
    public static void main(String[] args){
        new StressTest().startTest();
    }

    void startTest(){
        int n=12;
        for(int i=0; i<n; i++){
            TestThread t = new TestThread(this);
            t.start();
        }
    }

    synchronized public void returnTime(long t){
        totalTime+=t;
        System.out.println("total time taken is " + totalTime);
    }
}

class TestThread extends Thread{
    long timetaken = 0;
    public String serverHost = "vaioz";
    Socket thisSocket = null;
    InputStream is;
    OutputStream os;
    StressTest mainThread;

    public TestThread(StressTest t){
        mainThread=t;
    }

    public void run(){
        try {

//open socket connection

thisSocket = new Socket(serverHost, 1501);
if (thisSocket == null) { //check whether the connection is successfully established
    System.out.println("Cannot establish connection.\n");
} else {
    System.out.println("Connection is established to server.\n");
}

// get the input and output streams
is = thisSocket.getInputStream();
os = thisSocket.getOutputStream();

WBXMLOutputter encoder = new WBXMLOutputter();
Element rootElement = new Element("login_request");
rootElement.addAttribute("username", "ftc97");
rootElement.addAttribute("password", "pass");
Document login = new Document(rootElement);

Date start = new Date();
encoder.output(login, os);

WBXMLBuilder parser = new WBXMLBuilder();
Document reply = parser.build(is);

Date end = new Date();
mainThread.returnTime(start.getTime() - end.getTime());
is.close();
os.close();
thisSocket.close();

} catch (Exception e) {
    e.printStackTrace();
}

}
11.8 Appendix 8

DOC1

<pre>
<?xml version="1.0"?>
<!DOCTYPE XYZ [
<!ELEMENT XYZ ( CARD )+>
<!ELEMENT CARD (#PCDATA | INPUT | DO)>
<!ATTLIST CARD NAME NMTOKEN #IMPLIED>
<!ATTLIST CARD STYLE (LIST|SET) 'LIST'>
<!ELEMENT DO EMPTY>
<!ATTLIST DO TYPE CDATA #REQUIRED>
<!ATTLIST DO URL CDATA #IMPLIED>
<!ELEMENT INPUT EMPTY>
<!ATTLIST INPUT TYPE (TEXT|PASSWORD)"TEXT">
<!ATTLIST INPUT KEY NMTOKEN #IMPLIED>
<!ENTITY nbsp "&nbsp;">]
<!-- This is a comment -->
<XYZ>
<CARD NAME="abc" STYLE="LIST">
<DO TYPE="ACCEPT" URL="http://xyz.org/s"/>
Enter name: <INPUT TYPE="TEXT" KEY="N"/>
</CARD>
</XYZ>

Total size = 488

WBXML

01 01 6A 12 'a' 'b' 'c' 00 ' ' 'E' 'n' 't' 'e' 'r' ' ' 'n'
'a' 'm' 'e' ':' ' ' 00 47 C5 09 03 00 05 01 88 06
86 08 03 'x' 'y' 'z' 00 85 03 'r' 's' 00 01 83 04
01 83 04 86 07 0A 03 'N' 00 01 01 01

total size = 57
11.9 Appendix 9

```xml
<?xml version="1.0"?>
<!DOCTYPE XYZ [ 
<!ELEMENT XYZ (CARD)+> 
<!ELEMENT CARD (#PCDATA | BR)*> 
<!ELEMENT BR EMPTY> 
<!ENTITY nbsp "&#160;"> 
]> 
<XYZ> 
<CARD> 
X & Y<BR/> 
X &nbsp; = &nbsp; 1 
</CARD> 
</XYZ> 

Total size (bytes) = 171

WBXML
01 01 03 00 47 46 03 ' ' X ' ' & ' ' Y 00 05 03 ' ' 
X' 00 02 81 20 03 '=' 00 02 81 20 03 '1' ' 00 01 01 

Total size (bytes) = 33