University of Southern Queensland

Faculty of Engineering & Surveying

XML Based Online Traffic Information

A dissertation submitted by

Sim Lee Kheng Shirley

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Engineering (Computer System Engineering)

Submitted: October, 2004

Abstract

This project is to develop an online traffic information which provides drivers with online traffic conditions like accidents, roadworks and traffic jams. It is believed that XML is suitable since it is designed to describe and focus on data, especially structured data.

The Extensible Markup Language (XML), which is derived from Standard Generalized Markup Language (SGML), is a simple, flexible text format, Unicode-base meta-language: a language for defining markup languages. XML is not limited by any programming language, operating system or software vendor. By being platform independent, XML can provide means for achieving interoperability between different programming platform and operating systems.

The following outcomes have been achieved:

1. A graphical display section consists of map of Singapore with incident icon like traffic light spoilt, accident and road works. Each of these icons consist an XML document.
2. A search engine which requires the end-user to input parameters like road name and date. Results are then displayed from the search and the end-user clicks on the required link to display the XML document.

University of Southern Queensland

Faculty of Engineering and Surveying

ENG4111/2 *Research Project*

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Engineering and Surveying, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Engineering and Surveying or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Prof G Baker

Dean

Faculty of Engineering and Surveying

Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.
I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Sim Lee Kheng Shirley

00310333100

_____

Signature

_____

Date

Acknowledgments

I would like to take this opportunity to thank the following people for helping out in this project:

1. **DR. ZHOU, Hong** (Project Supervisor)- For giving her guidance in the area of Software Engineering techniques.
2. **DR. LEIS, John** (Project Supervisor) - For giving his guidance throughout the project itself.

Sim Lee Kheng Shirley

University of Southern Queensland
October 2004

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Chapter Overview

This chapter will look at how the idea of this project is develop and the aims of this project. It will also compare Extensible Markup Language(XML) and Hypertext Markup Language(HTML). It will show how the decision to use XML came about. Next it will show the software product available and the decision to use which software product to implement this project.

1.2 Rationale

Internet has been an information ground for many people. Information about weather, currency exchange, maps, technologies and many more can be found on the web. Currently, there are web sites containing traffic information in United States and Canada. All this web site is using HTML and Active Server Page (ASP). But currently there is no such web site in Singapore. Real-Time Online Traffic Information will help drivers to know the current status on the road. If these drivers avoid those roads that have accident, it will prevent traffic jams and can also prevent another accident. These are the reasons why this project is develop.

1.3 Project Aims

The "XML-Based Online Traffic Information" system develop in this project aims to render traffic information through a website to the end-user. The web site will eventually produce two types of applications for the rendering of traffic information. The first application, which is a very simple one, was done using a graphical display. The graphical display which will show a Singapore map, which has icon put on certain areas to signify the activity going on in that area. The end-user just has to click on the icon he wants and a small dialog box containing the XML document will appear showing the activity happening in that area. This XML document will have a style sheet link to it.

The second application uses a search-engine which requires the end-user to input certain parameters like road name and date. Results are then displayed from the search and the end-user clicks on the required link to display the XML document. This XML document will also have a stylesheet link to it.

Even if it is displayed using the browser's default stylesheet, which actually shows raw XML data, the end-user will definitely still be able to interpret the document, as XML is actually a content markup language.

1.4 XML vs HTML

The only particular difference about this project is that the markup of the document is to be done in XML rather than HTML. The term "XML" stands for eXtensible Markup Language. This is a new and upcoming language that has the capability of having customised tags and content to its markup. Customizing of tag is not allow for HTML, as it is merely a markup language for display. Customizing its tags will require much more complex programming.

The best way to compare and contrast how XML and HTML represent data is to take a look at data represented using each of this language. Listing 1.1 shows some data representation in HTML.

Listing 1.1: A Basic HTML Code

```
<!-- The original html recipe -->
<html>
<head>
<title>Example</title>
</head>
<body>
<h3>Example</h3>
<h4>Authors</h4>
<table border="1">
<TR BGCOLOR="#308030"><TH>Name</TH><TH>Nationality</TH></TR>
<TR><TD>Victor Hugo</TD><TD>French</TD></TR>
<TR><TD>Sophocles</TD><TD>Greek</TD></TR>
<TR><TD>Leo Tolstoy</TD><TD>Russian</TD></TR>
<TR><TD>Alexander Pushkin</TD><TD>Russian</TD></TR>
<TR><TD>Plato</TD><TD>Greek</TD></TR>
</table>
</body>
</html>
```

Figure 1.1 shows how it looks in a browser.

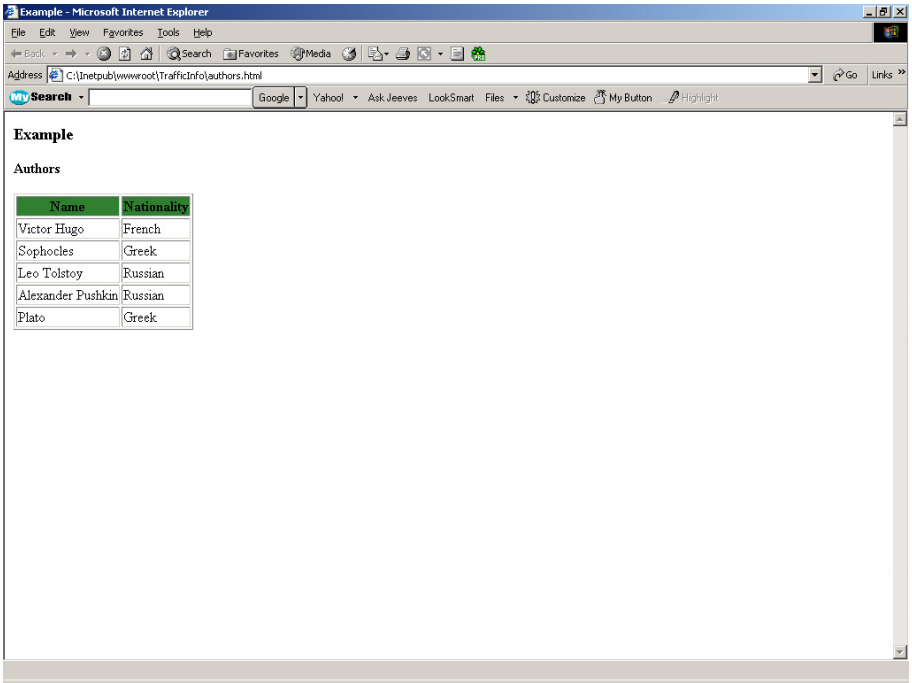

Figure 1.1: A Basic HTML Page

There are some positive aspects on how HTML represent its data. It is readable in the browser form, it can be displayed in any browser and a cascading sheet can be used for further control on the formatting. However, there is one big negative aspect that cover all the positive aspects. There is nothing in the codes to indicate meaning of any of

its elements. The data contain has no context. A program may scan and pick up data but it would not know what kind of data it represent (Coleman 2001).

For this example, assumption was made that the first column is name and second column is nationality. If the formatting is change, the whole application will fall apart.

The problem can be seen more in depth by attempting to extract the data and store it in a database. Since semantic information was taken out when it was translated into HTML, this information has to be resupply in order to store it meaningfully in a database. In other words, data has to be translated back to HTML when data needs to be extract out of the database. This is because HTML is not a suitable storage medium for semantic information (Tamura 2000).

What happen if the data is represented in XML? This is show in Listing 1.2.

Listing 1.2: Simple XML Code

```
<?xml version='1.0'?>
<authors>
  <author>
    <name>Victor Hugo</name>
    <nationality>French</nationality>
  </author>
  <author period="classical">
    <name>Sophocles</name>
    <nationality>Greek</nationality>
  </author>
  <author>
    <name>Leo Tolstoy</name>
    <nationality>Russian</nationality>
  </author>
  <author>
    <name>Alexander Pushkin</name>
    <nationality>Russian</nationality>
  </author>
  <author period="classical">
    <name>Plato</name>
    <nationality>Greek</nationality>
  </author>
</authors>
```

The tags in Listing 1.2 relate to authors, not using formatting like HTML. The file still remains readable, so it retains the simplicity of the HTML format, but the data

now has context. A program that parses this file will know exactly who Victor Hugo is.

Most users who had use XML finds that XML is actually more human readable than HTML. And XML accomplishes the goal of being at least as simple to use as HTML, yet it is orders of magnitude more powerful. It explains the information in an author in terms of authors, not in terms of how to display authors. Formatting of XML document will be discuss in the later chapter.

1.5 Software Selection

Microsoft has made XML an important initiative and has even mentioned it in a television commercial. Lotus is placing heavy emphasis on XML and is devoting major development resources to this technology. Other companies including IBM, Oracle, and Sun are likewise placing special emphasis on XML (Tamura 2000).

For SUN Microsystem, it is known as Java Technology XML. It contain the subcategories (SUN 2004):

→ Java Architecture for XML Binding (JAXB)

→ Java API for XML Processing (JAXP)

→ Java API for XML Registries (JAXR)

→ Java API for XML-Based RPC (JAX-RPC)

→ SOAP with Attachments API for Java (SAAJ)

As for Apache, it is known as Apache XML Project. Currently it consists of a number of sub-projects, each of it focused on a different aspects of XML (Apache 2004):

→ *Xerces* - XML parsers in Java, C++ (with Perl and COM bindings)

→ *Xalan* - XSLT stylesheet processors, in Java and C++

→ *FOP* - XSL formatting objects, in Java

→ *Forrest* - XML/XSLT project community websites

→ *Xang* - Rapid development of dynamic server pages, in JavaScript

→ *SOAP* - Simple Object Access Protocol

→ *Batik* - A Java based toolkit for Scalable Vector Graohics (SVG)

→ *Axis* - A Java based implementation of SOAP

→ *Commons* - A meta-project of common XML-oriented code

→ *Security* - Java and C++ libraries for encryption and signature functions

Microsoft has setup an XML Developer Center to aid programmer. The center consist of information on XML Technology, XML downloads, code samples and how to build XML application. These application are created using Microsoft product.

Therefore after much research on XML, it was decided that the project use Microsoft's products. This is due to Microsoft's speed in the development of XML and also due to product compatibility. This can be seen from Microsoft XML Developer Center.

The only limitation in adopting this approach is that since the application was done using Microsoft's Active Server Page Technology and Frontpage, it might not be able to work compliantly if a Netscape browser were to be used. As such the client must have Microsoft's Internet Explorer 5.0 and above if he/she wants to view the document.

More information on the design and details of this project can be referenced in the later parts of this report.

1.6 Specific Objectives

This project seeks to develop online traffic information. This system provides the drivers with online traffic conditions like accident, roadwork and traffic jam. Therefore it is believed XML is suitable since it is design to describe and focus on data, especially structured data.

1.7 Overview

This dissertation is organized as follows:

Chapter 2 Web Languages: This chapter will look into the history of web languages and the various languages used for this project.

Chapter 3 System Requirement & Installation: This chapter will show the system required and show how the various software components are installed.

Chapter 4 Development of the Web Site: This chapter will look more in depth the development of the project.

Chapter 5 Software Development Life Cycle: This chapter shows in detail the software development methodology used.

Chapter 6 Conclusion & Further Work: The project will be concluded in this section, future work would be discuss and also states the shortcomings of this project.

Chapter 2

Web Languages

2.1 Chapter Overview

This chapter examine the various languages for creating web pages. The history of web languages and how XML came about. It will also look into the components needed for XML document, and languages like Active Server Page(ASP) and Javascript, and the web server used. All the above is necessary to build the XML-Based Online Traffic Information.

2.2 History of Languages

2.2.1 Early Markup Languages

Early markup languages started with IBM, who pursued the idea of a standard method for structuring document with the goal of facilitating the exchange and manipulation of data, about 40 years ago. The first generation of markup language was Generalized Markup Language (GML) and it was used only internally in IBM to create various kinds of documents. Although there were similar technologies developed at other organizations but these were all proprietary and incompatible with each other. Furthermore there was no worldwide standardization (Rose 2000).

Generalized Markup Language was developed further and become the first standardized markup language, Standard Generalized Markup Language (SGML). SGML was expanded to be suitable as an all-purpose markup standard, and it was soon being used in a wide range of settings. In 1996, SGML was released as an official standard by the International Organization for Standardization (ISO).

The official SGML specification is over 150 very technical pages. It covers many special cases and unlikely scenarios. It is so complex that almost no software has ever implemented it fully. Programs that implement or rely on different subsets of SGML are often incompatible. The special feature that one program considers essential is all too often considered extraneous fluff and omitted by the next program. Nonetheless, experience with SGML taught developers a lot about the proper design, implementation, and use of markup languages for a wide variety of documents. Much of that general knowledge applies equally well to XML. SGML is an extremely powerful and flexible technology, which unavoidably entails a great deal of complexity and processing overhead (Rose 2000).

One thing which is very clear is that XML documents are not just used on the Web. XML can easily handle the needs of publishing in a variety of media, including books, magazines, journals, newspapers, and pamphlets. XML is particularly useful when you need to publish the same information in several of these formats. By applying different stylesheets to the same source document, you can produce web pages, speaker's notes, camera-ready copy for printing, and more.

2.2.2 Hypertext Markup Language

Further development of the markup language was motivated by the Internet. Many documents of various types like text, graphics were available on the Internet. Tim Berners-Lee, who was a software engineer at the European Laboratory for Particle Physics in Switzerland at that time, realized that access to these documents would be vastly improved if they could be linked to one another in a meaningful way that would enable user to move easily between related documents. A method had to be developed for marking up these documents in order to specify the links between documents and

also to specify how a document was to be display in the browser. The resulting language was a subset of SGML called Hypertext Markup Language (HTML). The birth of World Wide Web, which came with HTML, consists of the entire web of linked documents. HTML became the standard language of the Web and almost any Web page on the Internet uses HTML (Coleman 2001).

Although HTML has been a great success but as the Web grew, developers wanted to include more and more things on the Web pages such as animations and databases. HTML, which was originally designed as a hyperlink and to display documents, was not up to the task. It was clear that HTML was not enough to meet the needs of Web developers. They need something more powerful and flexible. The most serious limitation of HTML is its fixed tag set. Web developers can only use the tags defined in HTML therefore there is no extensibility. On the other hand, SGML fully support custom tags. But the complexity and processing overhead of SGML made it unsuitable for general Web use (Coleman 2001).

2.2.3 Standard Organization

The XML standard, which was created by the World Wide Web Consortium (W3C), is an open public organization. Its task is to develop technologies and standards for the Internet. In addition to XML, W3C is the force behind standards for Hypertext Markup Language (HTML), Portable Network Graphics (PNG, a graphics file format for Web use), and HyperText Transfer Protocol (HTTP, the standard for information transmission on the Internet), to name a few. Although XML standard is public, it is not owned or dominated by any single commercial interest (Quin 2004).

2.3 Extensible Markup Language

In 1996, World Wide Web Consortium (W3C) set the goal to develop a standard that would provide the power and flexibility of SGML in a form that was suitable for use on the Web. XML actually is a subset of the Standardized General Markup Language (SGML). SGML is an internationally accepted standard for describing just about any

type of information. However, as mention in the earlier section, it's way too complex for the relatively simple world of the web. Therefore, the W3C created a modified version of SGML specifically for the web, named it XML, and released it on an unsuspecting public sometime in 1998 (Vaswani 2002). Among the requirement for the new standard, it includes three of SGML's most significant benefits:

Extensibility The developer can define own custom tags

Structure The language syntax follows a well-defined structure

Validation Documents can be validated against a data model

The W3C committee worked on its task for almost 2 years. In February 1998, it was ready for the first version of the standard for the new language. It is known as Extensible Markup Language version 1.0, it remains as one of this writing the current XML specification, or in W3C terms, the XML Recommendation. XML is a markup language that is designed to describe data. It matches the meaning to data in a document by using a meaningful tag name (Quin 2004).

One of the greatest misunderstandings of the Extensible Markup Language will replace HTML in web design. In actual fact, XML is an extension of HTML. While XML and HTML share similar characteristics, such as the use of tag to markup data, it is important to know that XML has a broader purpose than just for the Web. The difference is that HTML was designed specifically to format data for web browsers and, as such, is limited to a predefined set of tags and functions. By itself, XML does not do anything. It simply uses tags to describe the data in a document and allows the document author to use meaningful tags to describe the data that is stored. In addition, the document author has the freedom to use whatever tag name he deems fit, hence the name *extensible*. The only time where XML really get useful is when it is used in conjunction with applications that can understand XML.

2.3.1 Features

XML was designed to have the following features (Vaswani 2002):

Ease of use ⇒ Since it can contain descriptive tags, XML documents are easy to read and understand, even for users with little or no computer knowledge. It is also simple to create: a non-technical user is typically able to create an XML document in far less time than it takes to create a corresponding HTML document. This simplicity is perhaps XML's greatest selling point.

Formal structure ⇒ Although XML allow authors to name and used its own tags, it still does impose some formal structure on a document. But the tags must be named and nested correctly; opening tags must have corresponding closing tags; and namespaces must be defined wherever they are required. These rules ensure that every XML document meets some minimum expectations of structure and syntax, and make it easier for applications and processors to deal with XML data.

Internet-friendly ⇒ XML was designed to be used on the Internet, where it plays two very important roles.

Firstly, XML provides a toolkit that enables users to describe the huge amount of data floating around on the Internet. This immediately opens the door to better organization and classification of information on the web, more intelligent search engines, and new types of links between data.

Secondly, XML provides a standard mechanism for information exchange, encoding data in a format that is easily transmittable from one computer to another using existing Internet Protocols and transport mechanisms.

Wide application support ⇒ Since XML is easy to use and easy to move around, it is not hard to write an application that uses XML-encoded data. A number of XML parsers are available online, XML editors, validators, and similar tools are gaining market share, and most popular web browsers now support XML.

2.3.2 Processing XML

There are two aspects to processing XML. One aspect is the generation of XML from a data source. The original data can be from a Domino database, a relational database, text file, spreadsheet, or any other type of object. The resulting XML file can contain as much or as little data as the author like and depends on the application. If all the

original information in the XML file is not included, the author will not be able to perform a round trip export and import (Tamura 2000).

The second aspect of processing an XML file involves reading an XML file and then doing something with it. For example, reading an XML file and create documents in a relational database, read an XML file and create transactions in a financial accounting system, or even to read an XML file and display a multimedia presentation. And the list will go on (Tamura 2000).

2.4 Components of XML

2.4.1 Document Type Definition (DTD)

Document Type Definition (DTD) files provide the XML parser with information about the structure of your document. DTDs specify the name of the root node of your document tree, which is the name of your document type. It is used to specify the type of data that can be included in an element, the relative order and position of the elements and which elements can be nested into other elements. In addition to the root node or document type, DTDs also describe the other elements of the document and their attributes. Below shows the definition of the location of the external DTD using a relative URL address.

The External DTD:

External DTDs are useful for creating a common DTD that can be shared between multiple documents. Any changes that are made to the external DTD automatically updates all the documents that reference it (Coleman 2001).

2.4.2 Extensible Stylesheet Language (XSL)

The Extensible Stylesheet Language is a language that uses XML syntax. It provides a powerful mechanism to process XML files. There are two primary purposes

for XSL. The first is similar to the purpose for stylesheets in HTML. That is, to format documents for display to users by specifying font faces, font sizes, styles, and so forth (Griffith 2002).

The process of using XSL to change the format of data is known as *transformation*. This is the second purpose. XSL perform a function, which is supplying human-readable data, is as important as XML itself. Nothing is ever stored in a database without the expectation that it will be extracted and presented in some human-readable format one day. In fact, presenting readable data is the entire purpose of Internet (Griffith 2002).

Without XSL or any stylesheet, XML document will only be display as a raw data with its tags. XML provides these capabilities with XML documents for data and XSL stylesheets for formatting.

2.5 Other Web Languages

2.5.1 Active Server Page

The active server page which is also known as ASP, provide request object, response object and session. The request object will read request, the response object will write data to the client and as for the session object, ASP will provide an application object to hold data that persist through all session, throughout the lifetime of the application. ASP can hold both the template of HTML and some of the programming codes. In fact, any HTML page is an Active Server Page, simply just change the extension to .asp. The only different is that it is process through an ASP processor where the server sends all the URL request that has a .asp extension (Floyd 2002).

Generally, Active Server Pages (ASP) allow programmer to access middle-tier and backend applications and to generate content dynamically. In this regard, ASPs are similar to CGI (Common Gateway Interface) scripts, which can be found typically on most Web servers including Apache and IIS. More importantly, ASPs allow programmer to quickly create Web applications using Microsoft's Internet Information Server (IIS). ASP allows programmer to easily access powerful server-side components and large

databases through ODBC connections.

Individual server pages typically include scripting code. Virtually any scripting language can be used, provided a scripting engine has been installed on the server. The two languages most commonly associated with ASP are VBScript and JavaScript, Microsoft's version of ECMAScript. Both scripting languages come with the standard ASP installation. The scripting engines are installed when IIS is installed (Floyd 2002). This is one of the reason why IIS is chosen for this project.

Features (Floyd 2002):

1. An ActiveX component
2. Allows scripting code to be embedded into HTML pages
3. Comes with built-in scripting support for JavaScript or VBScript
4. Allows developers to generate HTML pages dynamically from the server
5. Can be coupled with other components that that allow programmer to connect to ODBC databases

2.5.2 JavaScript

JavaScript is a general-purpose programming language; its use is not restricted to web browsers. JavaScript was designed to be embedded within, and provide scripting capabilities for, any application. From the earliest days, in fact, Netscape's web servers included a JavaScript interpreter, so that server-side scripts could be written in JavaScript. Similarly, Microsoft uses its JScript interpreter in its IIS web server and in its Windows Scripting Host product, in addition to using it in Internet Explorer (Flanagan 2001).

It is a lightweight, interpreted programming language with object-oriented capabilities. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers and embellished for web programming with the

addition of objects that represent the web browser window and its contents (Flanagan 2001).

This client-side version of JavaScript allows executable content to be included in web pages which means that a web page need not be static HTML, but can include programs that will interact with the user, control the browser, and dynamically create HTML content. Syntactically, the core JavaScript language resembles C, C++, and Java, with programming constructs such as the if statement, the while loop, and the && operator.

The similarity ends with this syntactic resemblance, however. JavaScript is an untyped language, which means that variables do not need to have a type specified. The object-oriented inheritance mechanism of JavaScript is like those of the little-known languages. Like Perl, JavaScript is an interpreted language, and it draws inspiration from Perl in a number of places, such as its regular expression and array-handling features (Flanagan 2001).

2.5.3 Web Server: Internet Information Service (IIS)

Microsoft's Internet Information Services (IIS) is the second most popular web server on the Net today. Although it is second most but there is not many high-quality options for adding XSLT support as there are for Apache (Harold 2003).

Microsoft publishes an unsupported XSL ISAPI Filter. Like other ISAPI filters, it sits between the requests and the file system, making changes as documents are requested. In particular, it transforms documents according to an XSLT stylesheet before forwarding them to the client that made the request. It can cache stylesheets and apply different stylesheets to match the browser or XML document type. This filter also supports pipelining of stylesheets so that the output of one transformation can be become input to the next. It can also transform ASP-generated content, as well as static XML files. It's based on the MSXML parser/XSLT engine and thus shares that engine's bugs (Harold 2003).

No matter which server or what language is needed in that project, there will definitely have a server-side XSLT plug-in for you. It is far more reliable to use this plug-in to

transform the documents on the server where the environment can be controlled by the programmer than to send the XML document and stylesheet to the client and hope it has the necessary software to transform the document itself. Since XSLT is a fairly processor- and memory-intensive process, this can place a significant load on the server. But this should not be preventing the programmer to use XML and separating content from presentation or taking advantage of the full power of XSLT (Harold 2003).

2.6 Chapter Summary

This chapter, by describing the history, syntax and components of XML, helps to lay a foundation for the rest of this project dissertation. It can be seen that the purpose of XML is to store data in a form that can be easily read and analyzed. It is quite common to use XML to store data and use the descriptive XML tags to specify how it should be displayed, but this is not inherent part of XML. And it is also very common to write applications that convert XML data into HTML for display.

Chapter 3

System Requirements & Installation

3.1 Chapter Overview

This chapter will show the system required and steps to install these various software. Installation is a very important step of a software project. Once the installation part is wrong, then the project will not work. Some of this codings need to be run in a editor program to see that it works before putting inside the web. So it is important to have these editor program install.

3.2 System Requirement

System Requirement	Usage
Microsoft 98 and above	Operating System
Microsoft Access	Used to setup the database.
Internet Explorer 5.0 or above	used as the browser and the validating parser which is embedded in it, was used to check the validity of the XML documents.
XML Writer	It is an editor that was used to write the XML documents, DTDs and the stylesheets using eXtensible Style Language (XSL).
Microsoft's Web Server, Internet Information Services (IIS) Or Personal Web Server	It was used to publish all the files for the application.

Table 3.1: System Requirement

System Requirement	Availability
Microsoft Access and Internet Explorer 5.0	They are included inside the Microsoft Office CD
XML Writer	It is a freeware and can be download at http://www.xmlwriter.com /download/download.shtml
Internet Information Services (IIS) Or Personal Web Server	It is included in the Windows 2000 Professional components It is included in the Windows 98 component

Table 3.2: Availability of System

3.3 Installation Procedure

3.3.1 Setting Up Database

The Steps are:

1. Select Administrative Tools from Control Panel
2. The Data Sources (ODBC) was double-clicked and the ODBC Data Source Adminstrator was accessed.
3. System DSN was chosen on the menu list and the Add button was clicked.
4. Select the driver for the data source.
5. At this step, the database was selected by clicking on the Select button. The Data Source Name was named TrafficInfo. This will set the database.

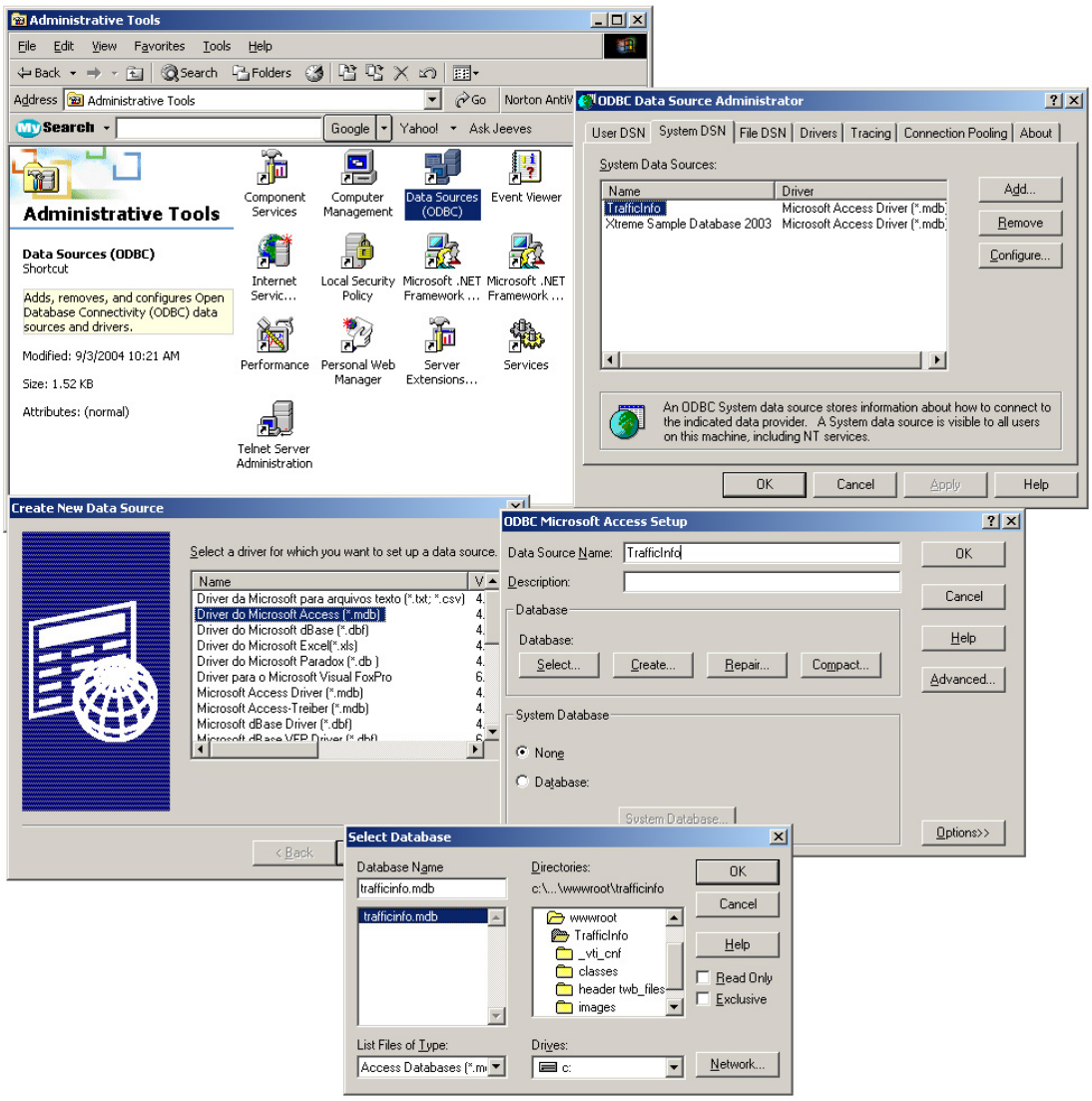

Figure 3.1: Setting up Microsoft access database.

3.3.2 Setting Up Internet Information Service

For the web server, only Internet Information Service will be demonstrate since it is used here.

The steps for installing are as follows:

1. Select Add/Remove Programs from Control Panel
2. Select Add/Remove Windows Components

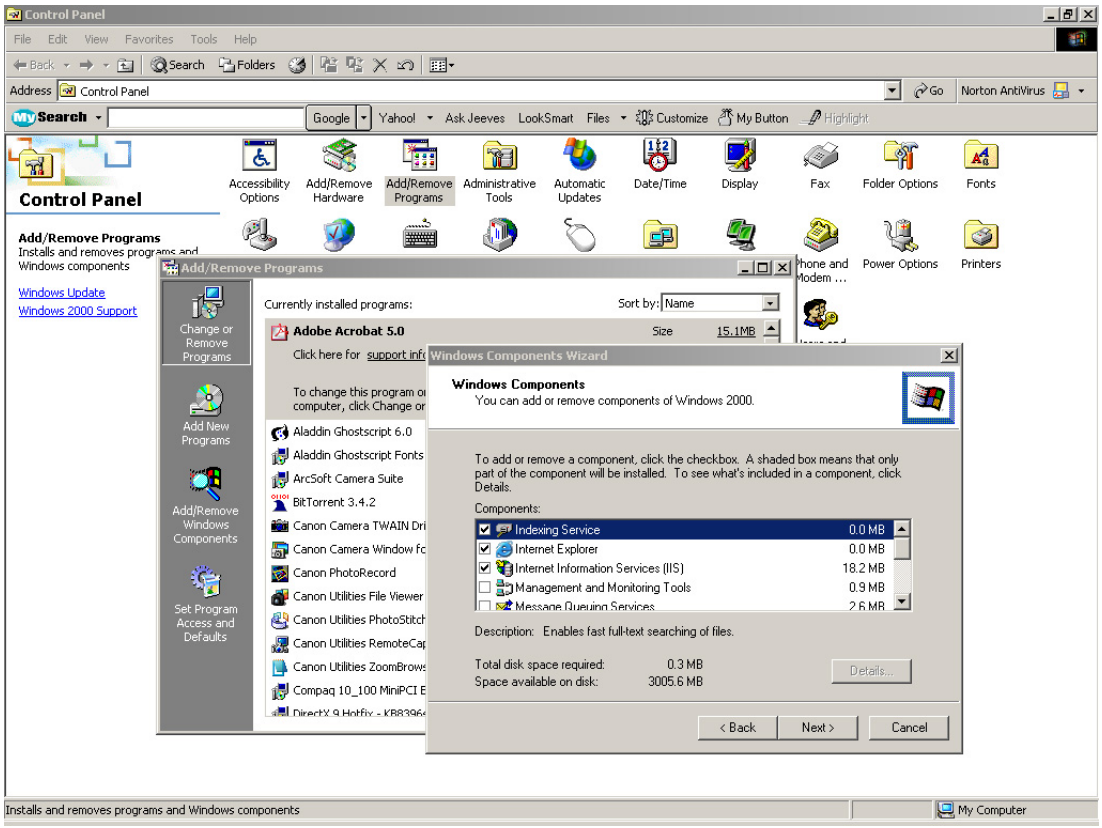

Figure 3.2: Setting up Internet Information Service(IIS).

3. In the Windows Components Wizard, check the Internet Information Service(IIS), then click next

4. And the IIS will be installed

After installing, the working directory would be in c:\Inetpub\wwwroot.

3.4 Chapter Summary

This chapter has describe the system requirement and gave the installation instruction for this project. Figure 3.1 and Figure 3.2 show how these installation take place.

Chapter 4

Development of the Web Site

4.1 Chapter Overview

This chapter will show how the following is created and develop in detail:

- ↪ Extensible Markup Language(XML) Document
- ↪ Document Type Definition (DTD)
- ↪ Extensible Stylesheet Language(XSL)
- ↪ Microsoft Access Database
- ↪ Javascript

4.2 Extensible Markup Language (XML)

XML files must begin with an XML declaration. Currently, the only version is 1.0, but the declaration must specify the version number so that documents using future versions can be distinguished from version 1.0. Below show the declaration of an XML document.

<?xml version="1.0" standalone="no"?>

Specifies the version of the XML standard that the XML document is conforms to. For standalone, use 'yes' if the XML document has an internal document type definition (DTD). Use 'no' if the XML document is linked to an external DTD, or any external entity references. In this project, an external DTD was used. Therefore, 'no' was chosen (Tamura 2000).

Rules for XML Declaration:

1. XML declaration must be in the first line of the document.
2. If there is any external entities, standalone must be 'no'.
3. Declaration of XML must be in lower case except for encoding declaration
4. XML declaration does not has an end tag like </?xml?>.

Example of XML syntax:

<age>12 years old </age>

Rules for XML Syntax:

1. The document must have a start tag followed by an end tag.
2. The start tag and end tag must have the same name.
3. The end tags contain a forward slash character in front of the tag name, to distinguish them from the start tags.
4. The markup in the xml document must conform to the requirements in the DTD.

4.3 Document Type Definition (DTD)

4.3.1 The <!DOCTYPE> Declaration

The <!DOCTYPE> Declaration The <!DOCTYPE> declaration describes the XML document type. It defines the root element, which is the parent element of all the other elements of the document. At most only one <!DOCTYPE> can be declaration within an XML file and therefore there is at most one root element. The declaration specified can be either inline or it can be in an external file (or both).

<!DOCTYPE Traffic SYSTEM "traffic.dtd">

A <!DOCTYPE> declaration is optional and is required only if the document need to be validate by the XML processor. An XML document is consider ***well-formed*** if it confined to all the rules of XML. It is ***valid*** if there is a <!DOCTYPE> declaration associated with the XML document and the document confines to the declaration. All the DTD declarations are required only if the XML documents need to be validate.

There is need to validate document from an unknown source. Validating the document will enable the XML processor to check the types of elements, their attributes, and the element nesting to ensure that they conform to a DTD. But if the document is taken from a known source, it is best to skip this validation part since validation takes time within process receiving the XML.

The only way to validate the document would be to provide own DTD, as shown in Listing C.12 and add the <!DOCTYPE> declaration to the XML document before processing (Tamura 2000).

4.3.2 The <!ENTITY> Declaration

The <!ENTITY> Declaration defines entities. There two types of entities: general entities and parameter entities. The use of these two types of entities depends on where and when the substitution of the entity occurs. General entities are defined in the DTD and can be used in the XML file. To declare a general entity, you use the following syntax:

<ENTITY entityname "entityvalue">

In the syntax description, entityname refers to the entity name and entityvalue refers to the entity's value. Here are some general entity declaration examples:

<ENTITY copw "Copyright (c) 2000">

To refer to a general entity, simply use an ampersand (&) before the entity name and a semicolon (;) following the name. When the XML file is parsed, the general entity will be replaced with its specified value. One common use for general entities is to specify special characters that are otherwise part of the XML syntax (Tamura 2000).

4.3.3 The <!ELEMENT> Declaration

<age> 22 </age>

This is an example of a single element with a start tag of <age>, an end tag of </age>, and content of 22! In general, an element consist of all the content from the start tag through the end tag. A tag can be just a part of an element.

The <!ELEMENT> type declaration is used to specify the name of the element and its valid content. Here is the syntax:

<!ELEMENT Name contentspec>

The element Name is case sensitive. The contentspec can be one of the two keywords *EMPTY* or *ANY* or a list of character data and/or children. This is shown in Listing C.12.

4.3.4 The <!ATTLIST> Declaration

The <!ATTLIST> declaration is used to specify an element's attribute list. Look at the syntax for <!ATTLIST> declaration below:

<!ATTLIST viewentries %root.attrs; toplevelentries %integer;#IMPLIED>

The first token after the keyword ATTLIST is the name of the element associated with the attribute list. This is the specification for the <viewentries> element's attributes. In this case notice that root.attrs is a parameter entity, which means that it will be substituted when the DTD is parsed. This entity is used so that any XML element that is a root element can contain a set of common attributes. The definition of the integer parameter entity is defined as CDATA (Tamura 2000). Therefore, after substitution, the attribute declaration is equivalent to line 5 of Listing C.12.

4.4 Extensible Style Language (XSL)

Refering to Listing C.11 for this section.

The XSL stylesheet begins with the XML declaration, just like any XML file. Following the header is the first XSL element,

<xsl:stylesheet>

Notice that a colon follows a prefix of xsl. This convention is called a namespace, and XSL uses a separate namespace so that its commands can be separated from any output that might be generated. Namespaces are described in detail on the Web at http://www.w3.org/TR/REC-xml-names/ (Rucker 2002).

The "xmlns:xsl" attribute is used to specify the version of the XSL transform specification that should be used. For this case, version 1.0 is used. The definitions for two template rules are next defined in the XSL file. Template rules are the key to the transformation of XML documents because they specify how the transformation should take place (Rucker 2002).

Template Rules

Several template rules can exist within a stylesheet, each template rule normally contains a match attribute. The match attribute is used to define the context in which the template rule applies. That is, template rules apply only in certain contexts, and the match attribute informs the XSL process when the particular template rule is appropriate (Tamura 2000).

Conceptually, when processing an XML document, it is first read and parsed into memory. The resulting parsed document is specified as a tree. The XSL stylesheet instructions are then processed, with the XML tree as input. The result of processing the XML input and the XSL instructions is another tree, which can then be output in a variety of formats. Because user control the output format, in addition to XML or HTML, he/she can write the output in any format he/she desire. The match attribute of a template rule specifies which nodes of the input XML tree apply to the template rule.

If the template rule applies to a particular node, the instructions within the template rule are used to construct the output. The parsed XML tree contains both a root node and a root element. The root node is considered the top of the tree. The only

content of the root node is a single child, which is the root element of the XML tree (Tamura 2000).

This is the first template in the XSL stylesheet:

<xsl:template match="/"><xsl:apply-templates/></xsl:template>

The match attribute with a single slash matches the root node (not root element) of the XML tree. Because it matches this root node, the XSL processor automatically processes it. It is analogous to the main program in Java. At any point in the processing, one node of the input tree is considered to be the current node. When processing starts, the current node is the root node.

The template that best matches the current node is selected and processed. Processing typically involves recursively matching and processing subsidiary nodes of the current node. Because the single slash matches the root node, the template is selected and processed.

The <xsl:apply-templates/> element is then processed. This instruction tells the XSL processor to recursively process any of the children of the current node. The root node has just one child, which in this is the <Record> element node. The XSL processor will look for any templates that match the <Record> element.

Listing 4.1: XSL Sample Template

```
......
.......
........
<xsl:template match=``Record">
<html>
<head></head>
<xsl:for-each select=``TrafficRecord">
<tr>
<td>
        <xsl:for-each select=``RoadName">
        <xsl:apply-templates/>
        </xsl:for-each>
</td>
</tr>
</html>
</xsl:template>
......
```

. . . . . .

In the template above, the match attribute indicates that the template rule applies when the node is a <Record> element. Following the beginning tag in Listing C.11, there's a set of HTML tags. These tags are sent directly to the output without modification.

<xsl:value-of select="RoadName"/>

The <xsl:value-of> element is used to obtain the value of the expression specified in the select attribute. If there is a "@" symbol, it is used to indicate that the name refers to an attribute name, not a tag name (Tamura 2000).

Note that because of the parsing of the original input and the processing of trees, the output nodes and line breaks can appear differently from the input.

Pattern Matching

Patterns are very important to transformations because patterns are used to search for a particular context. They are used to select attributes and also to match attributes. The use of the / pattern, which matches the root node (not the root element).

An example of pattern matching is shown below

<xsl:template match="Traffic/Record">
<tr><td>Document created</td><td>
<xsl:value-of select="."/></td></tr>
</xsl:template >

This code fragment will match the following fragment:

<Traffic><Record> 12345 </Record></Traffic>

The value of the <xsl:value-of> element will be the number that is found within the <Record> element.

The <xsl:attribute> element creates an attribute on the most recent start tag element.

It will work only if no output has been generated for the tag yet (Tamura 2000).

4.5 Microsoft Access Database

Understanding Access Component

There are several types of objects that make up an Access database. The ones that most nonprogrammer or nonprogrammer wants is the ability to create or modify include tables, forms, queries, and reports (Bruck 2002).

Table ⇒ The matrix of rows and columns that contains the data in the database, as well as the information about data properties, formatting, and validation rules for fields.

Form ⇒ Used to enter and edit information in the Access database. Forms can be extensively customized to facilitate the data entry process, and can also contain data validation rules and formatting information for entry fields.

Query ⇒ Extracts selected data from one or more database tables, and presents it in a table format. However, the query doesn't actually contain data as what a table does, it is dynamically generated each time it is run. Queries can also display summary information or grouped information.

Report ⇒ Presents data from one or more tables or queries, and is generally used for printed output. Access provides tools for formatting reports. Other database objects include data access pages (used to publish Access data on Web pages), macros (to automate database use), and modules (more sophisticated automated procedures built with Visual Basic). In addition, a switchboard is a special type of form used to present menus for using Access applications. Switchboards are created automatically when you create an Access database using a wizard.

Creating Access Database

1. Open Access, and choose General Templates from the task pane.

2. In the Databases tab of the Templates dialog box, choose the template that most resembles the database you need to create and click OK.

3. As you go through the steps of the wizard, the tables (business or personal) of the database are preset and cannot be changed during the database creation process. After the sample fields in that particular table has been selected to be used in the new table, the fields can be rename using the "rename" icon. This is shown in Figure 4.1. Rename those fields to Incident, Information, Road and Acdate.

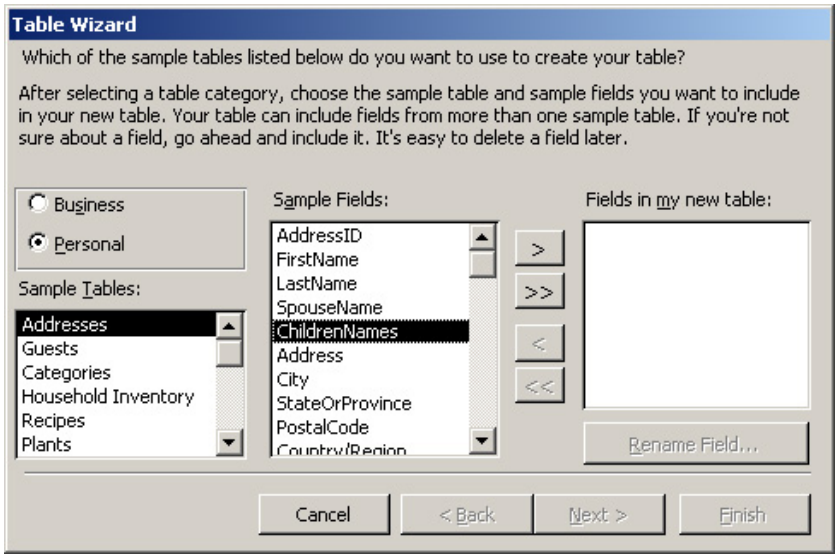

Figure 4.1: This show how a database is created.

4.6 Javascript

Client-side JavaScript code can be embedded within the HTML or ASP documents in a number of ways:

1. Between a pair of <script> and </script> tags

2. From an external file specified by the 'src' attribute of a <script> tag

3. In an event handler, specified as the value of an HTML attribute such as onclick or onsubmit

4. As the body of a URL that uses the special javascript: protocol

The last technique will not be since it is not used in this project. The following section will show the allowed structure of JavaScript programs on the client side.

4.6.1 The <script> Tag

Client-side JavaScript scripts are part of an ASP file and are coded within <script> and </script> tags. Any number of JavaScript statements can be place between these tags; these statements are executed in order of appearance, as part of the document loading process. <script> tags may appear in either the <head> or <body> of an HTML or ASP document. This is shown in Listing C.8.

A single HTML document may contain any number of non-overlapping pairs of <script> and </script> tags. These multiple, separate scripts are executed in the order in which they appear within the document. While separate scripts within a single file are executed at different times during the loading and parsing of the HTML file, they constitute part of the same JavaScript program: functions and variables defined in one script are available to all scripts that follow in the same file (Flanagan 2001). Take for example:

<script> var x = 1; </script>

Later in the same HTML page, this 'x' can be referred to, even though it's in a different script block. The context that matters is the HTML page, not the script block:

<script>document.write(x);</script>

The document.write( ) method is an important and commonly used one. When it is used as above, it inserts its output into the document at the location of the script. When the script finishes executing, the HTML parser resumes parsing the document and starts by parsing any text produced with document.write( ).

Although JavaScript is by far the most commonly used client-side scripting language, it is not the only one. In order to tell a web browser what language a script is written in, the <script> tag has an optional language attribute. Browsers that understand the specified scripting language run the script; browsers that do not know the language

ignore it (Flanagan 2001).

For Javascript, it is written in:

<script language="JavaScript"> // JavaScript code goes here </script>

This is also shown in Listing C.8. VBScript can also be used as the scripting language. The only browser that supports VBScript is Internet Explorer, so scripts written in this language are not portable. VBScript interfaces with HTML objects in the same way that JavaScript does, but the core language itself has a different syntax than JavaScript (Flanagan 2001). VBScript is not discuss in detail here.

JavaScript is the default scripting language for the Web, and if the language attribute is omitted, both Netscape and Internet Explorer will assume that the scripts used are written in JavaScript.

The HTML 4 specification standardizes the <script> tag, but it deprecates the language attribute because there is no standard set of names for scripting languages. Instead, the specification prefers the use of a type attribute that specifies the scripting language as a MIME type. Thus, in theory, the preferred way to embed a JavaScript script is with a tag that looks like this:

<script type="text/javascript">

In practice, the language attribute is still better supported than this new type attribute.

The HTML 4 specification also defines a standard and useful way to specify the default scripting language for an entire HTML file. If JavaScript is the only scripting language used in a file, simply include the following line in the <head> of the document:

<meta http-equiv="Content-Script-Type" content="text/javascript">

In this way, JavaScript scripts can be used without specifying the language or type attributes.

Since JavaScript is the default scripting language, there is really not a need to use the

language attribute to specify the language in which a script is written. However, there is an important secondary purpose for this attribute: it can also be used to specify what version of JavaScript is required to interpret a script. When you specify the language="JavaScript" attribute for a script, any JavaScript-enabled browser will run the script. However, if a script that uses the exception-handling features of JavaScript 1.5, used the following tag to avoid syntax errors in browsers that do not support this version of the language (Flanagan 2001):

<script language="JavaScript1.5">

4.6.2 Including Javascript files

As of JavaScript 1.1, the <script> tag supports a src attribute. The value of this attribute specifies the URL of a file containing JavaScript code. It is used like this:

<script src="../../javascript/util.js"></script>

A JavaScript file typically has a .js extension and contains pure JavaScript, without <script> tags or any other HTML.

A <script> tag with the src attribute specified behaves exactly as if the contents of the specified JavaScript file appeared directly between the <script> and </script> tags. Any code that does appear between these tags is ignored by browsers that support the src attribute Note that the closing </script> tag is required even when the src attribute is specified and there is no JavaScript between the <script> and </script> tags (Tamura 2000).

The followings are the advantages of using the src tag (Tamura 2000):

1. It simplifies HTML files by allowing the blocks of JavaScript to be remove.
2. When a function or other JavaScript code is used by several different HTML files, simply keep it in a single file and read it into each HTML file that needs it. This reduces disk usage and makes code maintenance much easier.
3. When JavaScript functions are used by more than one page, placing them in a

separate JavaScript file allows them to be cached by the browser, making them load more quickly.

4. When JavaScript code is shared by multiple pages, the time savings of caching more than outweigh the small delay required for the browser to open a separate network connection to download the JavaScript file the first time it is requested.

5. Because the src attribute takes an arbitrary URL as its value, a JavaScript program or web page from one web server can employ code such as subroutine libraries to be exported by other web servers.

4.6.3 Event Handlers

JavaScript code in a script is executed once, when the HTML file that contains it is read into the web browser. A program that uses only this sort of static script cannot dynamically respond to the user. More dynamic programs define event handlers that are automatically invoked by the web browser when certain events occur (Tamura 2000).

For example, when the user clicks on a button within a form. Because events in client-side JavaScript originate from HTML objects (such as buttons), event handlers are defined as attributes of those objects.

Another example, to define an event handler that is invoked when the user clicks on a checkbox in a form, simply specify the handler code as an attribute of the HTML tag that defines the checkbox:

<input type="checkbox" name="opts" value="ignore-case" onclick="ignore-case = this.checked;">

It is more important to know the function of the onclick attribute in the code above. The string value of the onclick attribute may contain one or more JavaScript statements. If there is more than one statement, the statements must be separated from each other with semicolons. When the specified event occurs on the checkbox, the JavaScript code within the string is executed. In the code above, the event is the click (Rucker 2002).

While any number of JavaScript statements are included within an event handler definition, a common technique, when more than one or two simple statements are required, is to define the body of an event handler as a function between <script> and </script> tags. Then this function can simply be invoked from the event handler. This keeps most of the actual JavaScript code within scripts and reduces the need to mingle JavaScript and HTML.

Some of the common used event handlers (Tamura 2000):

onclick ↪ This handler is supported by all button-like form elements, as well as <a> and <area> tags. It is triggered when the user clicks on the element. If an onclick handler returns false, the browser does not perform any default action associated with the button or link.

onchange ↪ This event handler is supported by the <input> , <select>, and <textarea> elements. It is triggered when the user changes the value displayed by the element and then tabs or otherwise moves focus out of the element.

onsubmit , onreset ↪ These event handlers are supported by the <form> tag and are triggered when the form is about to be submitted or reset. They can return false to cancel the submission or reset. The onsubmit handler is commonly used to perform client-side form validation.

4.7 Chapter Summary

This chapter shows how the codes are develop for each language used. There are examples given to aid the understanding of these codes. The outlook of the web page and the source codes can be found in Appendix B and C respectively.

Chapter 5

Software Development Life Cycle

5.1 Chapter Overview

This chapter will look into the software life cycle model used in this project. It will show that it is important to use the model in developing project, be it small or larger scale project. Software Life Cycle is actually a map that guides those involve in a project to move forward and helps them to understand whether they have reached their destination. Software Life Cycle will develop the project in a few phases, each with a sequence of activities. These sequence may not be linearly sequential because they may flow from one another, repeat themselves or run concurrent.

5.2 Software Life Cycle Model

A software process framework or skeleton, describe what is to be perform in each phase of a project development via the activities of each phase. The phase used here refers to the distinguishable stage in the development process. Life Cycle phases represent distinct and successive periods with entry and exit criteria (Futrell 2002).

In the olden days of software development, codes were mostly and then debug without any detail planning. Without any formal design and analysis , it is impossible to know

which direction you are going or even know where is the destination. There is no way to access requirement or whether the quality criteria has been satisfy.

Therefore, during the early phase, a framework for development phase need to be formalize, placing emphasis on up-front requirements and design activities, and on producing documentation (Futrell 2002).

5.3 Waterfall Life Cycle Model

The application of the Waterfall Model should only be limited to situations in which the requirement and the implementation of those requirements are well-understood. Waterfall Model performs well for product cycles with a stable definition and well understood technical methodologies. The critics of the model in Figure 5.2 must admit that the modified version of the waterfall is far less rigid than the original, Figure 5.1. This includes the iteration of phases, concurrent phases and change of environment.

The reverse arrows in Figure 5.1 allow for iterations of activities within phases. To reflect concurrency among phases, the rectangles are often stacked or activities within the phases are often listed beneath the rectangle showing concurrence. Although the modified Waterfall Model is much more flexible than the classic, it is still not the best choice for rapid development project (Rucker 2002).

5.3.1 Development

The execution of Waterfall Model starts at the upper left of Figure 5.2 and progress through the orderly sequence of steps. For pure Waterfall Model, it is assumed that each of the subsequent phase will only begin when the activities of the current phase have been completed. But for Waterfall Model with feedback like Figure 5.2, it allows the possibility of revisiting the earlier stages. This is better because it is impossible to completely specific and plan the whole project in advance of writing codes.

There is a define entry (input) and exit (output) criteria for each phase. Internal

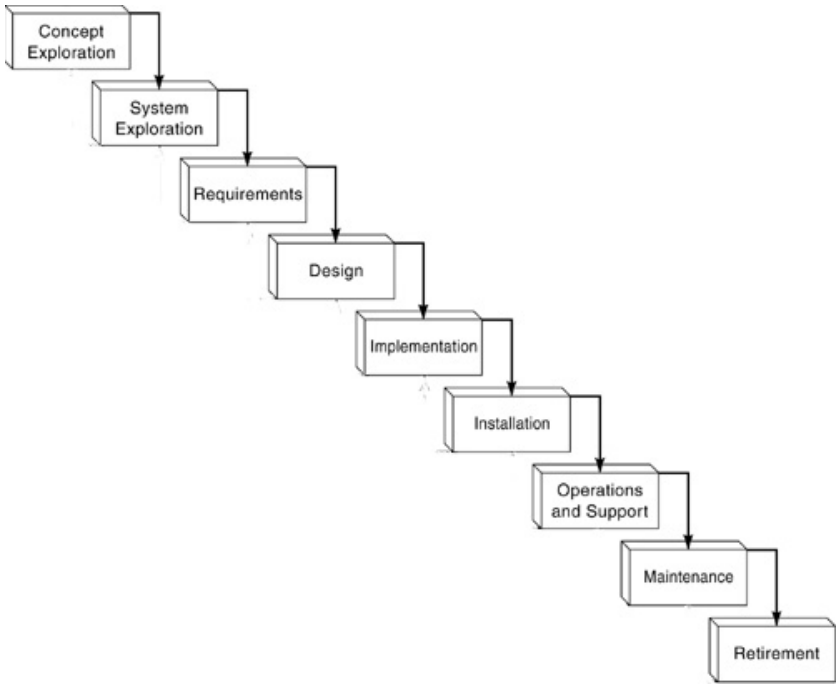

Figure 5.1: Pure Waterfall Life Cycle Model

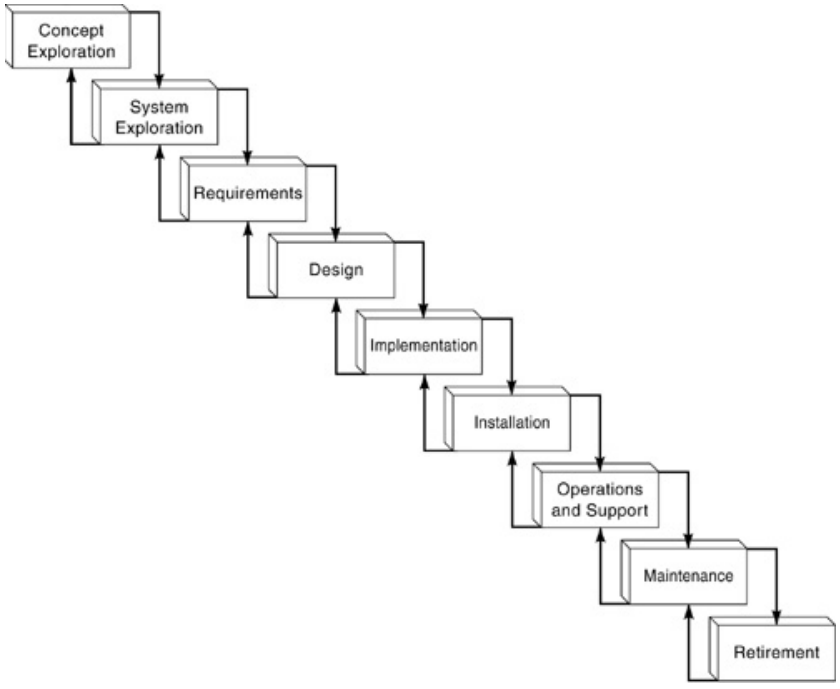

Figure 5.2: Waterfall Life Cycle Model with feedback (Rucker 2002).

or external deliverables are the output from each phase including documentation and software. The transition from one phase to another is by passing formal review. It is a way to provide customers an insight into the development process and to check on the product stability. Therefore, the passing of formal reviews indicates an agreement that the phase has ended and the next phase can begin (Futrell 2002).

5.3.2 The Phases

The description of the phase activities below is based on Figure 5.2 and show how these phases is related in the development of this project.

Concept Exploration Examine the requirements at the system level and to determine feasibility. ⇒ Refer to Chapter 1 under Section: Rationale.

System Allocation This maybe skipped for software only system. For system that require the development of both the software and hardware, the required functions are mapped to the software or hardware based on the overall system architecture. ⇒ Refer to Chapter 3 under Table 3.1.

Requirement This defines the software requirements for the system's information domain, function, behaviour, performance and interfaces. ⇒ Refer to Chapter 3 under Table 3.2.

Design Develop and represents a coherent, technical specification of the software system, this includes the data structure, software software architecture, interface representation and procedural (algorithm) detail. ⇒ Refer to Chapter 4.

Implementation This phase results in the transformation of the software design description to a software product. This produce the source codes, database, and documentation constituting the physical transformation of the design. For product which is a purchased application package, the major implementation activities are the installation and testing of the software package. For software product, the major activities are programming and testing codes. ⇒ Software design description refer to Chapter 1 under Section: Project Aims and Software Selection. Source Code refer to Appendix C.

Installation This involved the installation of software, check out and product to be formally accepted by the customer. ⇒ Refer to Chapter 3 under Section: Installation

Operation & Support This involve the user operation of the system and the ongoing support which includes providing technical assistance, consulting with the user, recording user requests for enhancements and changes, and also to handle corrections or errors.

Maintenance This phase is concerned with the resolution of the software errors, faults, failure, enhancements, and changes generated by the support phase. It consist of the iterations of development and support feedback of anomaly information.

Retirement This phase is to remove an existing system from its active use, by either terminating its operation or replacing it with a new system or an upgrade version of the existing system. ⇒ This phase is not used in this project because it not removing an existing system from its active use.

Integral This involve project initiation, project monitoring and control, quality management, verification and validation, configure management, documentation development and training throughout the entire life cycle.

5.3.3 Advantages & Disadvantages

The advantages are as follows:

This refers to project that is well-suited for Waterfall Model. Waterfall Life Cycle Model is well-known by non-software customers and end-users. It tackle complexity in an orderly way and works well with project that are well-understood but still complex. It is easy to understand with simple goal to complete the required activities. It provides requirement stability and also provides template into which methods for analysis, design, code, test and support can be placed. Waterfall Model defines quality control procedures. Each deliverable is reviewed as it is completed. The quality of the system is determine by procedures. The milestones are well understood. It is easy to track the

progress of the project using a timeline or Gantt chart. The completion of each phase is used as a milestone (Futrell 2002).

The disadvantages are as follows:

This refers to project that is not well-suited for Waterfall Model. The inherently linearly sequential nature does not allow any attempted to go back 2 or more phases to correct a problem or deficiency which would result in major increases in cost and schedule. It does not reflect the problem solving nature of the software development. The phases are tight rigidly to activities, and is not how the people or teams really work (Futrell 2002).

5.4 Chapter Summary

The concept of Software Development Life Cycle is important in software design. The phases in the life cycle is for software designer to follow in order to ensure that their software is design according to the client request and that the dateline is meet. This project follows this life cycle as shown in the development

Chapter 6

Conclusions and Further Work

As shown in the previous chapter, the basis aims of the XML-Based Online Traffic Information has been achieved. However, several enhancements are possible. These will be briefly discuss in this final chapter.

6.1 Future Work

6.1.1 Additional Features

Other than just an interactive map and a search engine, real-time images on the road condition can be used. This is similar to the one shown in the news in Singapore, where traffic condition on major roads are shown when news reporter are reporting the traffic condition on the road.

This can be liaised with the local Land Transport Authority as they will have a set of camera to monitor the traffic conditions on the road. In this way, the drivers will have a few more choices to know the traffic condition on the road.

6.1.2 WAP Site

As mode of communication changes with time, many people nowadays carry mobile phone around. And the technology of mobile phone are still ongoing. Users can log onto the WAP site to got information just like on a web site. They can even check email and information like weather, news and many more. Therefore, mobile phone would be a better choice to have an online traffic information. The followings are the reason why this is suggested:

↪ Mobile Phone is smaller and easier to carry around than a laptop. This would lead to a real-time online traffic information service for drivers.

↪ Almost everyone has one mobile phone

6.1.3 Taxi Drivers Communicator

Currently, the taxis in Singapore have a communicator installed inside. This replace the radio communication device which they used to communicate with the control center. Online-based traffic information can also be develop to view in the communicator. This will enable taxi drivers to know the road condition better since the communicator follow wherever they go.

6.2 Shortcomings

There are two main shortcomings for this project:

1. This Online-Based Traffic Information web page can only be view in Internet Explorer 5.0 or later. Other web browser like Mozilla, Netscape and Opera might not present this web page in the way that it should be.

2. This web page can only be viewed on the Internet, drivers might not bring a laptop or desktop with them all the time, therefore it is only useful if the drivers are leaving from home or office. This is the reason why the future work is suggested.

6.3 Achievement of Project Objectives

The following objectives have been addressed:

The Use of Latest Web Technology Chapter 1 show how XML, the latest web technology, is used in this project.

Comparison of XML & HTML Chapter 1 has compare and contrast these two web languages. Examples are given.

Understanding of Web Languages Chapter 2 shows the birth of web languages and how XML came about.

Uses of various languages Chapter 4 present how XML, XSL, DTD, ASP and JavaScript are combine to develop this project.

Search Engine & Interactive Map Chapter 4 show how this is achieved.

Importance of System Requirements Chapter 3 show the system requirements and how some of the requirement affect this project.

Importance of Software Development Life Cycle Chapter 5 show how software development life cycle does to software design.

Phases of Waterfall Life Cycle Chapter 5 shows the life cycle that is used for the development of this project. The phases of Waterfall Life Cycle are follow in this project.

Further Enhancement Chapter 6 present a number of enhancement of this project which can be undertaken if this project is to be develop further.

References

Apache, X. P. (2004), *Apache XML Project*.
`http://xml.apache.org/`.

Bruck, B. (2002), *Taming the Information Tsunami*, Microsoft Press.

Coleman, P. (2001), *XML Complete*, Sybex, California.

Flanagan, D. (2001), *JavaScript: The Definitive Guide, 4th Edition*, O'Reilly.

Floyd, M. (2002), *Special Edition Using XSLT*, Que.

Futrell, R. T. (2002), *Quality Software Project Management*, Prentice Hall PTR.

Griffith, A. (2002), *Java, XML and JAXP*, John Wiley & Sons Inc, Canada.

Harold, E. R. (2003), *Effective XML: 50 Specific Ways to Improve Your XML*, Addison Wesley.

Quin, L. (2004), *Extensible Markup Language (XML)*.
`http://www.w3.org/XML`
current April 2004.

Rose, G. (2000), *XML: A Primer*, M & T Books, United Kingdom.

Rucker, R. (2002), *Software Engineering & Computer Games*, Addison Wesley.

SUN, M. (2004), *Java Technology XML*. `http://java.sun.com/xml/`.

Tamura, R. A. (2000), *Domino 5 Web Programming with XML, Java, and JavaScript*, Que.

Vaswani, V. (2002), *XML and PHP*, New Riders Publishing.

Appendix A

Project Specification

University of Southern Queensland

FACULTY OF ENGINEERING & SURVEYING

ENG 4111/4112 Research
PROJECT SPECIFICATION

FOR:	SIM LEE KHENG SHIRLEY
TOPIC:	XML-Based Online Traffic Information
SUPERVISORS:	Dr. Hong Zhou Dr. John Leis
ENROLMENT:	ENG4111- S1, X, 2004; ENG4112- S2, X, 2004
PROJECT AIM:	This project seeks to develop an online traffic information. This system provides the drivers with online traffic conditions like accident, roadwork and traffic jam. Therefore it is believed XML is suitable since it is design to describe and focus on data, especially structured data.
PROGRAMME:	**Issue A, 26th February 2004**

1. Research on the various web languages used for creating web pages.
2. A database where all the different parameters such as date, type of incidents, linked XML document, stylesheet are stored for retrieval.
3. An application where necessary inputs are received from the end-user and processed to give the required output. This will be written in Javascript.
4. Stylesheets are linked to the XML document to present the XML document in a more desirable way.
6. XML documents with the inclusion of an external Document Type Definition (DTD).
7. All of the above components are linked together and placed in Microsoft Personal Web Server.
8. A map of Singapore, which includes icons on it. Different icons will represent different type of accidents.
9. Each icon will link to an XML document and will be displayed on a pop-up window.

AGREED:
_____ (Student) 4 MAY 2004
_____, _____ (Supervisors) 12/03/2004 / /

Appendix B

User View of the XML-Based Online Traffic Information

B.1 XML-Based Online Traffic Information

The following screen shots show how the XML-based online traffic information works.

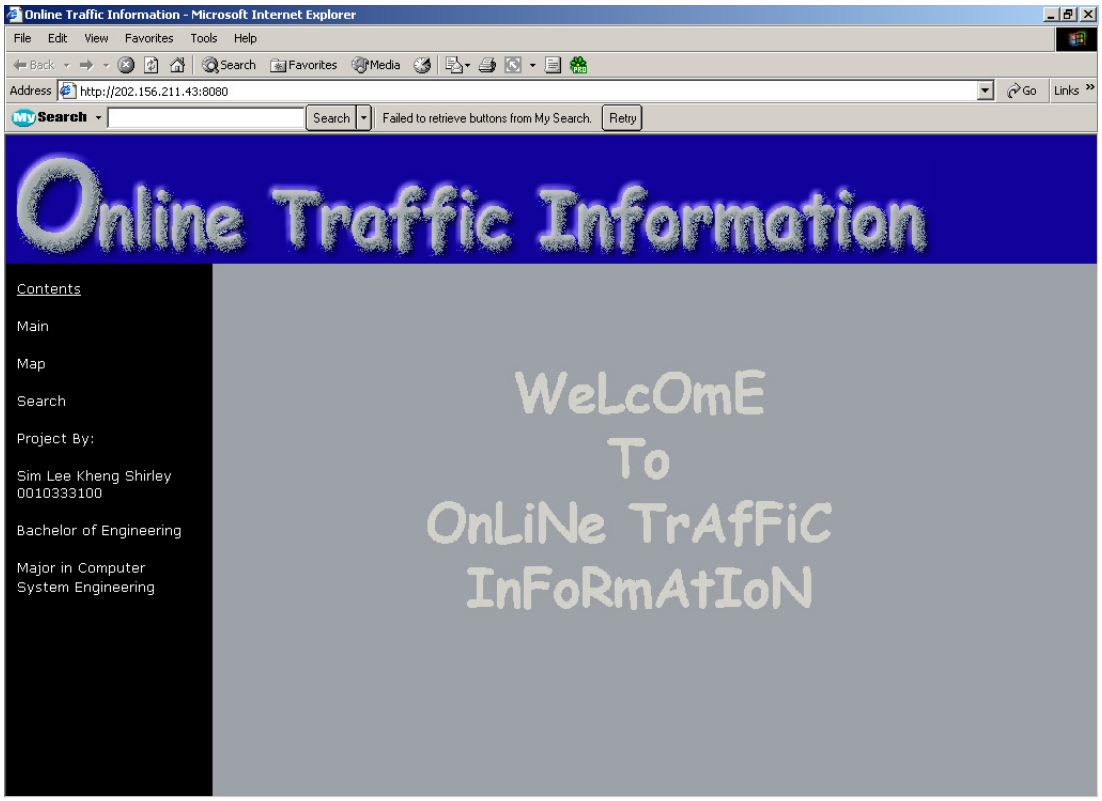

Figure B.1: This is the main page.

Figure B.2: When the "map" on the side menu is click, this page is shown.

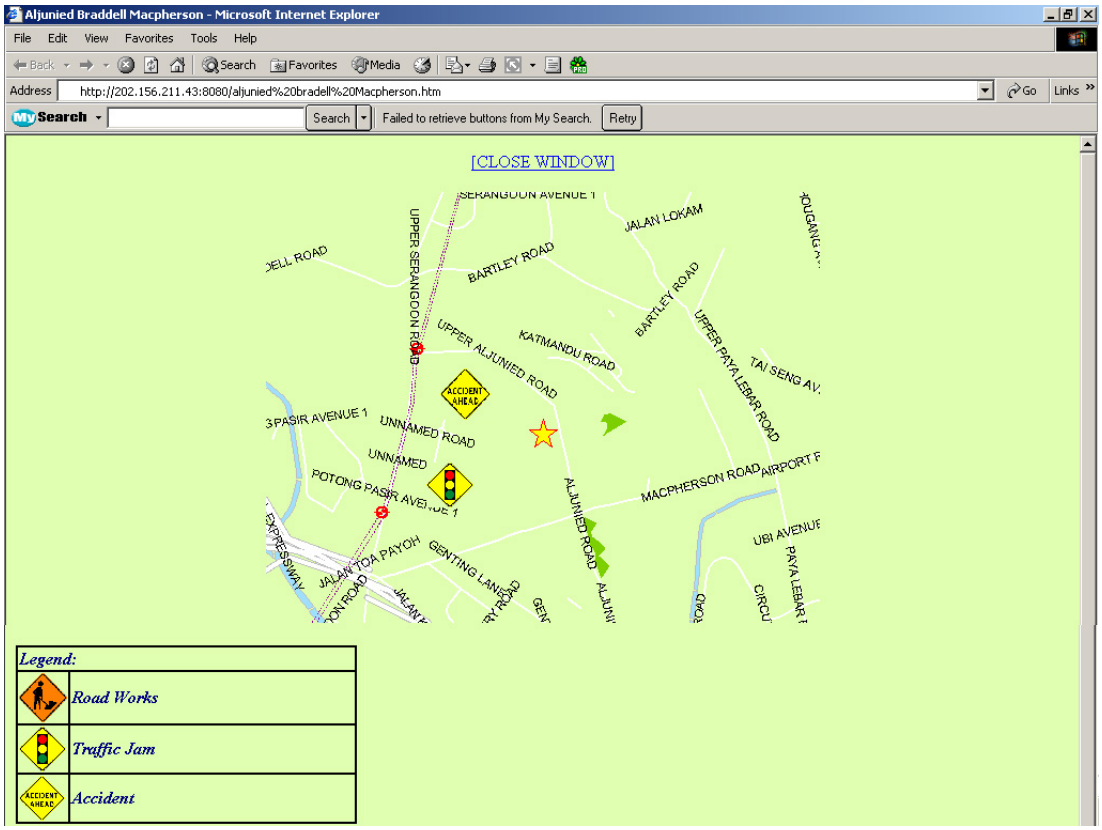

Figure B.3: When a town/suburban is click, the map of that town/suburban appear next. For this example, Aljunied/Braddell/Macpherson is used.

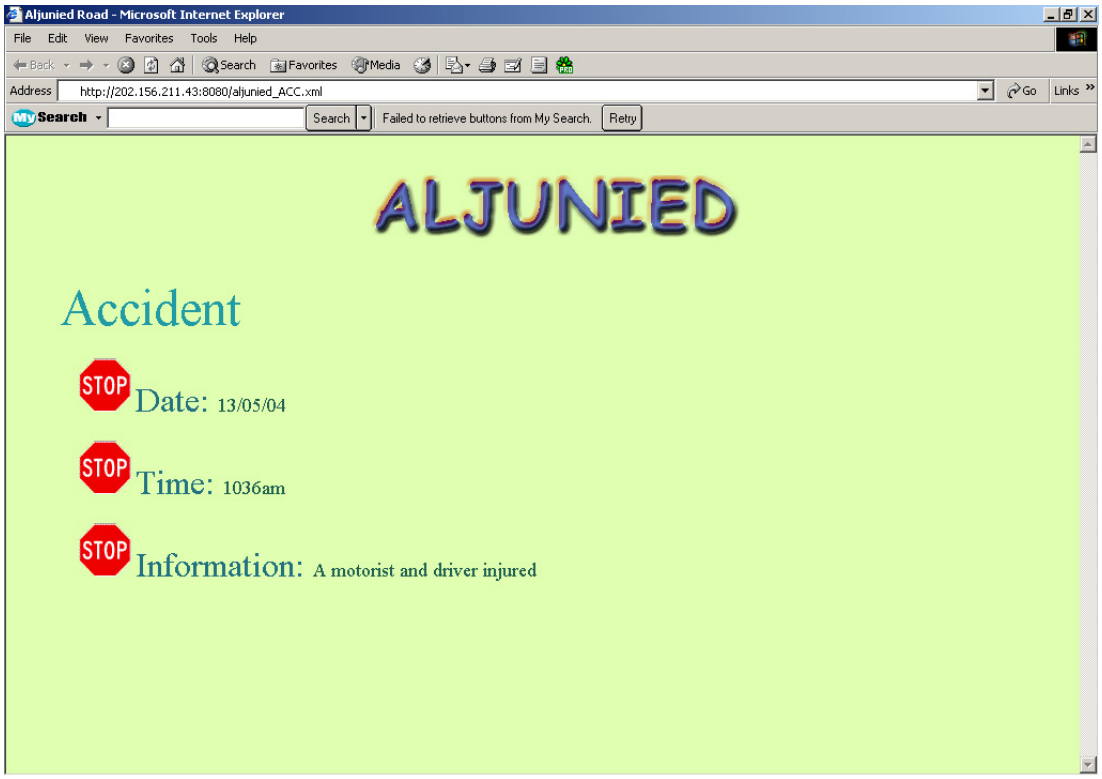

Figure B.4: This is the XML document after the incident icon is click.

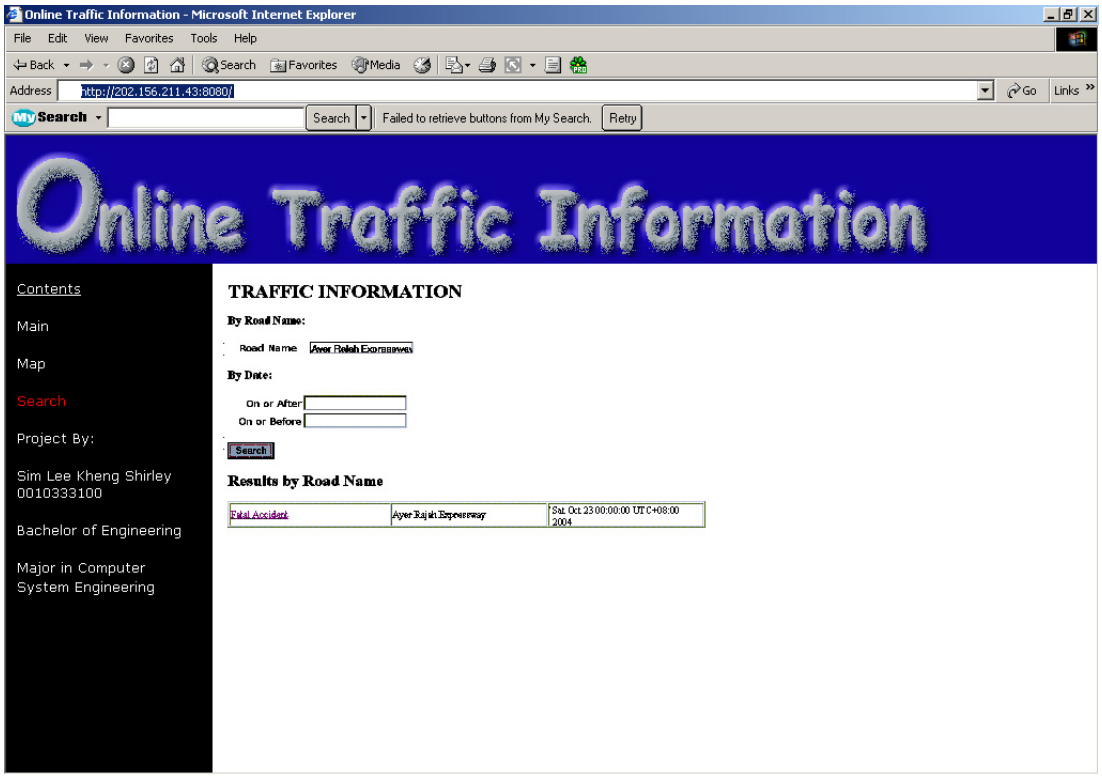

Figure B.5: This page appear when "search" is click from the main page. After filling up the parameter, click the "search" icon. If the parameter is found, a result link is shown.

Figure B.6: This show the result of the search. It is an XML document.

Appendix C

Source Code

Listing C.1: Default Page for Traffic Information

```
<html>

<head>
<meta http-equiv="Content-Type" content="text/html;
        charset=windows-1252">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<title>Online Traffic Information</title>
</head>

<frameset framespacing="0" border="0" rows="120,*"
                                          frameborder="0">
  <frame name="banner" scrolling="no"
           noresize target="contents" src="header.htm">
  <frameset cols="193,*">
    <frame name="contents" target="main" src="sidemenu.htm">
    <frame name="main" src="main.htm" target="_self">
  </frameset>
  <noframes>
  <body>

  <p>This page uses frames, but your browser doesn't
                                              support them.</p>

  </body>
  </noframes>
</frameset>

</html>
```

Listing C.2: Side Menu for Main Page

```
<html>

<head>
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html;
        charset=windows-1252">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<title>Contents</title>
<base target="main">
<style>
body {font-family: verdana, helvetica, sans-serif;
        font-size:12pt;
        background-image: url('url');
        background-attach: scroll;
        background-position:  left top;
        background-repeat: no-repeat
        }
p{font-family: verdana, helvetica, sans-serif;
        font-size: 10pt;
        color: #ffffff;
        }
dt{font-family: verdana, helvetica, sans-serif;
        font-size: 9pt;
        color: #663300;
        }

td{font-family: verdana, helvetica, sans-serif;
        font-size: 8pt;
        }

a{text-decoration:
        none;}
a:hover{text-decoration: regular;
        color: #ff0000;
        }

</style>
</head>
<body bgcolor="#000000" link="#ffffff" vlink="#ffffff"
                                        alink="#ff0000">
<p align="left"><u>Contents</u></p>
<p align="left"><a href="main.htm">Main</a></p>
<p align="left"><a href="main_map.htm"
                        target="_blank">Map</a></p>

<p align="left"><a href="traffic.htm"
                        target="main">Search</a></p>
```

```
<p align="left">Project By:</p>
<dl>
   <font color="#FFFFFF"><font size="2">
                        Sim Lee Kheng Shirley</font>
   </font><font size="2">
   <font color="#FFFFFF">0010333100</font> </font>
   <p>
   <font color="#FFFFFF"><font size="2">
                    Bachelor of Engineering</font>
   </font><font size="2">
   </font></p>
   <font size="2" color="#FFFFFF">
           Major in Computer</font><font size="2">
   </font>
   <font size="2" color="#FFFFFF">
                          System Engineering</font>
</p>
</dl>
</body>
</html>
```

Listing C.3: Main.htm of Default Page

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1252">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<title>Main</title>
<base target="_self">
<style>
body {font-family: verdana, helvetica, sans-serif;
        font-size:12pt;
        background-image: url('url');
        background-attach: scroll;
        background-position:  left top;
        background-repeat: no-repeat
        }
p{font-family: verdana, helvetica, sans-serif;
        font-size: 10pt;
        color: #ffffff;
        }
dt{font-family: verdana, helvetica, sans-serif;
        font-size: 9pt;
        color: #663300;
        }

td{font-family: verdana, helvetica, sans-serif;
        font-size: 8pt;
        }

a{text-decoration:
        none;}
a:hover{text-decoration: regular;
        color: #ff0000;
        }
</style>
</head>
<body bgcolor="#9DA2A8">
<p align="center">
 </p>
<p align="center">
 </p>
<p align="center">
<img border="0" src="images/welcome.jpg" width="459"
                                        height="249"></p>
<p align="center"> </p>
</body>
</html>
```

Listing C.4: Header.htm of Default Page

```
<html>

<head>
<meta http-equiv="Content-Type" content="text/html;
charset=windows-1252">
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<title>Traffic Information</title>
<base target="contents">
</head>

<body bgcolor="#10029A">

<p><img border="0" src="images/header.jpg"
        width="858" height="99"></p>

</body>

</html>
```

Listing C.5: Interactive Map

```
<html xmlns:v="urn:schemas-microsoft-com:vml"
          xmlns:o="urn:schemas-microsoft-com:office:office"
          xmlns="http://www.w3.org/TR/REC-html40">

<head>
<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<meta name="ProgId" content="FrontPage.Editor.Document">
<meta http-equiv="Content-Type" content="text/html;
        charset=windows-1252">



<title>Map of Singapore</title>
</head>
<body bgcolor="#A7DEEC">
<p align="center">
<img border="0" src="images/map of Singapore.jpg"
                    width="534" height="163"></p>
<p align="center">
<img border="0" src="images/singmap.gif" align="baseline"
                              width="588" height="344"></p>
<p align="center">
<b><font color="#FF9966" size="4">TOWNS IN SINGAPORE</font>
</b></p>
<p align="center">
<a href="aljunied braddell macpherson.htm"
          target="map of singapore" style="text-decoration:
        none">Aljunied / Braddel / Macpherson
          </a>
<a href="amk bishan thomson.htm" target="map of singapore"
          style="text-decoration: none">
          Ang Mo Kio / Bishan / Thomson</a>
<a href="bedok upper east coast.htm"
          target="map of singapore" style="text-decoration:
        none">Bedok / Upper East Coast</a></p>
<p align="center">
<a href="bt batok cck bt panjang.htm"
          target="map of singapore" style="text-decoration:
        none">
          Bukit Batok / Chua Chu Kang / Bukit Panjang</a>
<a href="bugis beach rd.htm" target="map of singapore"
      style="text-decoration: none">Bugis / Beach Road</a>
<a href="city hall clarkquay.htm" target="map of singapore"
      style="text-decoration: none">City Hall / Clark Quay
          </a></p>
<p align="center">
<a href="clementi west coast.htm" target="map of singapore"
          style="text-decoration: none">Clementi / West Coast
```

```
        </a>
<a href="dunearn newton.htm" target="map of singapore"
          style="text-decoration: none">Dunearn / Newton</a>
<a href="holland bt timah.htm" target="map of singapore"
          style="text-decoration: none">
          Holland / Bukit Timah</a>
<a href="hougang seng kang.htm" target="map of singapore"
          style="text-decoration: none">Hougang / Seng
Kan</a><a href="hougang seng kang.htm"
          target="map of singapore">g</a></p>
<p align="center">
<a href="jurong tuas.htm" target="map of singapore"
          style="text-decoration: none">Jurong / Tuas</a>
<a href="kaki bt eunos geylang.htm" target="map of singapore"
          style="text-decoration: none">
         Kaki Bukit / Eunos / Geylang</a>
<a href="lim chu kang.htm" target="map of singapore"
          style="text-decoration: none">Lim Chu Kang</a>
<a href="loyang changi.htm" target="map of singapore"
          style="text-decoration: none">Loyang / Changi</a>
         </p>
<p align="center">
<a href="mandai lentor.htm" target="map of singapore"
          style="text-decoration: none">Mandai / Lentor</a>
<a href="merah commonwealth.htm" target="map of singapore"
          style="text-decoration: none">
         Bukit Merah / Commonwealth</a>
<a href="orchard river valley.htm" target="map of singapore"
          style="text-decoration: none">
         Orchard / River Valley</a>
<a href="pasir ris tampines.htm" target="map of singapore"
          style="text-decoration: none">
         Pasir Ris / Tampines</a></p>
<p align="center">
<a href="serangoon jln besar.htm" target="map of singapore"
          style="text-decoration: none">
         Serangoon / Jalan Besar</a>
<a href="telok blangah.htm" target="map of singapore"
          style="text-decoration: none">Telok Blangah</a>
<a href="tpy kallang whampao.htm" target="map of singapore"
          style="text-decoration: none">
         Toa Payoh / Kallang / Whampoa</a>
<a href="upper bt timah.htm" target="map of singapore"
          style="text-decoration: none">Upper Bukit Timah</a>
         </p>
<p align="center">
<a href="woodlands kranji.htm" target="map of singapore"
          style="text-decoration: none">Woodlands / Kranji</a>
<a href="yck jln kayu.htm" target="map of singapore"
```

```
          style="text-decoration: none">
          Yio Chu Kang / Jalan Kayu</a>
<a href="yishun sembawang.htm" target="map of singapore"
          style="text-decoration: none">Yishun / Sembawang</a>
          </p>
<p align="center"> </p>
<p align="center">
          <a href="javascript:window.close()">[CLOSE WINDOW]</a>
          </p>
</body>

</html>
```

Listing C.6: Map of Town

```
<title>Map of Singapore</title>
<base target="_self">
<body bgcolor="#A7DEEC">

<p align="center">
<img border="0" src="images/singmap.gif"

width="588" height="344"></p>
```

Listing C.7: XML Document for Town of Aljunied

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE Traffic SYSTEM "traffic.dtd">
<?xml-stylesheet type="text/xsl" href="traffic1.xsl" ?>
<Traffic>
        <Record id="111">
        <TrafficRecord>
                <Incident>Accident</Incident>
                <Date>13/05/04</Date>
                <Time>1036am</Time>
                <Info>A motorist and driver injured</Info>
        </TrafficRecord>
        </Record>
</Traffic>
```

Listing C.8: ASP for Traffic Information

```
<%@ LANGUAGE = JavaScript %>

<%

    var dbConn, dbRecordSet;

    var parser = Server.CreateObject("MSXML.DOMDocument");

    parser.loadXML(Request.ServerVariables("QUERY_STRING"));

    if (parser.readyState == 4 && parser.parseError == 0)
    {
       var docroot = parser.documentElement;

       if(docroot.nodeName=="PaperQuery")
       {
          dbConn = Server.CreateObject("ADODB.Connection");
          dbConn.Open("TrafficInfo.mdb","","");
          dbRecordSet = Server.CreateObject("ADODB.RecordSet");

          AssembleQueries(docroot);

          dbConn.Close();
          dbConn = null;
          dbRecordSet = null;
       }
       else
         Response.Write("<InvalidQuery></InvalidQuery>");
    }
    else
       Response.Write("<Error></Error>");

    function AssembleQueries(oRoot)
    {
          var childNode;
          var sQueryStem = "SELECT Traffic.Incident,
                           Traffic.AcdDate, Traffic.RoadName,
                           Traffic.Information FROM Traffic ";

          Response.Write("<PaperResponse>");

          for (var nChild = 0; nChild < oRoot.childNodes.length;
                                                  nChild++)
          {
                  childNode = oRoot.childNodes.item(nChild);
                  if (childNode.nodeType == 1)
                  {
                          if (childNode.nodeName == "BYROAD")
```

```
                                    AssembleRoad(sQueryStem,
                                                childNode);

                            if (childNode.nodeName == "BYTIME")
                                    AssemblePubDate(sQueryStem,
                                                childNode);
                    }
            }
            Response.Write("</PaperResponse>");
    }


    function AssembleRoad(sStem, oChild)
    {
            var paramNode;
            var sQuery = "";
            var regExp, strFiltered

            if (oChild.childNodes.length > 0)
            {
                    paramNode = oChild.childNodes.item(0);
                    if (paramNode.nodeType == 1 &&
                            paramNode.nodeName == "RNAME")
                    {
                            regExp = /%20/g;
                            strFiltered =
                            paramNode.childNodes(0).
                            nodeValue.replace(regExp, " ");
                            sQuery =
                            sStem + "WHERE RoadName LIKE '"
                            +strFiltered + "';";
                            MakeResponse(1, sQuery);
                    }
            }
     }

     function AssemblePubDate(sStem, oChild)
     {
            var paramNode;
            var sQuery = "", sConstraintClause = "";

            var paramNode;
            var sBefore = "", sAfter = "";
            var sConstraintClause = "", sQuery = "";
            var nParam = 0;

            for (nParam = 0; nParam <
                    oChild.childNodes.length; nParam++)
            {
```

```
            paramNode = oChild.childNodes.item(nParam);

            if (paramNode.nodeType == 1)
            {
                if (paramNode.nodeName == "BEFORE")
                        sBefore = "AcdDate <= #" +
                        paramNode.childNodes(0).
                                    nodeValue+"#";

                if (paramNode.nodeName == "AFTER")
                        sAfter = "AcdDate >= #" +
                        paramNode.childNodes(0).
                                nodeValue + "#";
                }
        }

        sConstraintClause = sAfter;

        if (sConstraintClause != "" && sBefore != "")
        sConstraintClause = sConstraintClause + " AND ";

        sConstraintClause = sConstraintClause + sBefore;

        if (sConstraintClause != "")
        sConstraintClause = "WHERE " + sConstraintClause;

        sQuery = sStem + sConstraintClause + ';';
        MakeResponse(2, sQuery);
}

function MakeResponse(nQueryType, sQuery)
{
        var sTypeEndTag;

        dbRecordSet = dbConn.Execute(sQuery);

        if (dbRecordSet.EOF && dbRecordSet.BOF)
        Response.Write("<NoResults></NoResults>");

        switch (nQueryType)
        {
                case 1:
                        Response.Write("<BYROAD>");
                        sTypeEndTag = "</BYROAD>";
                        break;

                case 2:
                        Response.Write("<BYTIME>");
                        sTypeEndTag = "</BYTIME>";
```

```
                        break;
    }

    while (!dbRecordSet.EOF)
    {
        Response.Write("<Paper>");
        if (dbRecordSet("Incident") != "")
            Response.Write("<incident>"
               +dbRecordSet("Incident")+"</incident>");

        if (dbRecordSet("Information") != "")
            Response.Write("<information>" +
                dbRecordSet("Information") +
                    "</information>");

        if (dbRecordSet("RoadName") != "")
            Response.Write("<road>" +
                dbRecordSet("RoadName") + "</road>");

        if (dbRecordSet("AcdDate") != "")
            Response.Write("<AcdDate>" +
                dbRecordSet("AcdDate") + "</AcdDate>");

        Response.Write("</Paper>");
        dbRecordSet.MoveNext();
    }
    Response.Write(sTypeEndTag);

}

%>
```

Listing C.9: Search Engine for Traffic Information

```
<html>

<head>

<script LANGUAGE="JavaScript">

var parser = new ActiveXObject("MSXML.DOMDocument");

function OnSearch()
{
    var sQueryXML = "";
    if (RNAME.value != "")
        sQueryXML += makeByName(RNAME.value);

    if (AFTER.value != "" || BEFORE.value != "")
        sQueryXML += makeByAcdDate(AFTER.value,
        BEFORE.value);

    if (sQueryXML != "")
    {
        sQueryXML = "<PaperQuery>" + sQueryXML +
        "</PaperQuery>";
        parser.async = "false";
        parser.load("trafficinfo.asp" + sQueryXML);
        HandleResponse(document.all("results"));
    }
}

function makeByName(sRoad)
{
    var sQueryString = "<BYROAD>";

    if (sRoad != "")
        sQueryString += "<RNAME>"+sRoad+"</RNAME>";

        sQueryString += "</BYROAD>";

        return sQueryString;
}

function makeByAcdDate(sStart, sEnd)
{

    var sQueryString = "<BYDATE>";

    if (sStart != "")
        sQueryString += "<AFTER>"+sStart+"</AFTER>";
```

```
    if (sEnd != "")
        sQueryString += "<BEFORE>"+sEnd+"</BEFORE>";

        sQueryString += "</BYDATE>";

        return sQueryString;
}

function HandleResponse(oDIV)
{
    var docroot;
    var nChildCount;
    var sTable = "";

    oDIV.innerHTML = "";

    if (parser.readyState == 4 &&
                parser.parseError.reason == "")
    {
        docroot = parser.documentElement;

        if (docroot.nodeName == "PaperResponse")
        {
            for (nChild = 0; nChild <
                docroot.childNodes.length; nChild++)
            {
                sTable = HandleResults
                (docroot.childNodes.item(nChild));
                if (sTable != "")
                oDIV.insertAdjacentHTML
                                ("beforeEnd", sTable);
            }
        }

        else
            window.alert("Improper response type;
                check with server administrator.");
    }

    else
        window.alert("Parser not ready or response not
            well-formed." + "Ensure you are contacting an
            appropriate server.");
}

function HandleResults(node)
{
    var i;
    var sReply = "";
```

```
    var child;

    if (node.nodeType == 1) // an XML ELEMENT
                    {
    if (node.nodeName == "BYROAD")
          sReply = '<h2>Results by Road Name</h2>';
    if (node.nodeName == "BYTIME")
          sReply = '<h2>Results by Date</h2>';
    if (sReply != "")
    {
          sReply += '<table border="1" width="100%">';
          for (i = 0; i < node.childNodes.length; i++)
          {
             child = node.childNodes.item(i);
             if (child.nodeType == 1)
             {
               if (child.nodeName == "Paper")
                  sReply += MakeHTMLFromPaper(child);
               if (child.nodeName == "NoResults")
                  sReply += '<tr><td width="100%">
                           No papers found</td></tr>';
              }
           }
                sReply += '</table>';
     }

    return sReply;
}

function MakeHTMLFromPaper(paperNode)
{
          var i;
          var childNode;
          var sRow = "";
          var sIncident = "", sPubLink ="", sRoad, sAcdDate;

          for (i = 0; i < paperNode.childNodes.length; i++)
          {
             childNode = paperNode.childNodes.item(i);

             if (childNode.nodeType == 1)
             {
                if (childNode.nodeName == "incident")
                   sIncident =
                   childNode.childNodes(0).nodeValue;

                if (childNode.nodeName == "information")
                   sPubLink =
                   childNode.childNodes(0).nodeValue;
```

```
            if (childNode.nodeName == "road")
               sRoad =
               childNode.childNodes(0).nodeValue;

            if (childNode.nodeName == "AcdDate")
               sAcdDate =
               childNode.childNodes(0).nodeValue;
         }
      }

      sRow += '<tr><td width="34%">';
      if (sPubLink != "")
      {
            sRow += '<a href="' + sPubLink + '">';
            if (sIncident != "")
                  sRow += sIncident + '</a>';
            else
                  sRow += 'Untitled paper' + '</a>';
      }
      else
      {
            if (sIncident != "")
                  sRow += sIncident;
      }

      sRow += '</td><td width="33%">';
      if (sRoad != "")
            sRow += sRoad;
            sRow += '</td>';

            sRow += '<td width="33%">';
      if (sAcdDate != "")
            sRow += sAcdDate;
            sRow += '</td></tr>';

      return sRow;
}

</script>

<meta name="GENERATOR" content="Microsoft FrontPage 5.0">
<title>TRAFFIC INFORMATION</title>
<meta name="Microsoft Theme" content="none, default">
<meta name="Microsoft Border" content="none, default">
</head>

<body>
<H1>TRAFFIC INFORMATION</H1>
```

```
<h3>By Road Name:</h3>


<table border="0" width="41%">
  <tr>
    <td width="41%" align="right">
        <font face="Verdana">Road Name</font></td>
    <td width="59%">
    <INPUT name=RNAME style="HEIGHT: 22px;
        WIDTH: 177px" size="20"></td>
  </tr>
 </table>
<H3>By Date:</H3>

<table border="0" width="40%">
  <tr>
    <td width="39%"><p align="right">
        <font face="Verdana">On or After</font></p></td>
    <td width="61%"><input NAME="AFTER" size="20" ></td>
  </tr>
  <tr>
    <td width="39%"><p align="right">
        <font face="Verdana">On or Before</font></p></td>
    <td width="61%"><input NAME="BEFORE" size="20" ></td>
  </tr>
</table>

<p align="left">
<input TYPE="button" VALUE="Search" ONCLICK="OnSearch()"
       NAME="SearchBtn">
</p>

<p align="left">
</p>
<DIV ID="results"> 
</DIV>
</body>
</html>
```

Listing C.10: XML Document for Traffic Information

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE Traffic SYSTEM "traffic.dtd">
<?xml-stylesheet type="text/xsl" href="traffic.xslt"?>
<Traffic>
    <Record id="111">
        <TrafficRecord>
            <RoadName>Hougang Street 11</RoadName>
            <Date>2/1/2004</Date>
            <Time>1202</Time>
            <Info>2 girls and 2 boys killed</Info>
        </TrafficRecord>
    </Record>
    <Record id="222">
        <TrafficRecord>
            <RoadName>Ang Mo Kio Ave 3</RoadName>
            <Date>29/3/2004</Date>
            <Time>1202</Time>
            <Info>1 elderly and 1 motorist injured
            </Info>
        </TrafficRecord>
    </Record>
    <Record id="333">
        <TrafficRecord>
            <RoadName>Havelock Road</RoadName>
            <Date>4/5/2004</Date>
            <Time>1850</Time>
            <Info>2 women and 1 man injured</Info>
        </TrafficRecord>
    </Record>
</Traffic>
```

Listing C.11: XSL for Traffic Information

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<html>
<head></head>
<body background="␣">
<p align="center"><img
            src="images/incident␣in␣singapore.jpg"/></p>
<xsl:for-each select="/Traffic/Record/TrafficRecord">
  <ul>
    <font face="Comic␣San␣MS" size="6" color="#229CA6">
    <xsl:value-of select="RoadName"/>
    </font>
  </ul>
  <ul>
    <p align="left" style="margin-left:␣18">
    <img src="images/stop.jpg"></img>
    <font face="Comic␣San␣MS" size="4" color="#237381">
    Date:
    <font face="Comic␣San␣MS" size="2" color="#226351">
    <xsl:value-of select="Date"/></font>
    </font></p>
  </ul>
  <ul>
    <p align="left" style="margin-left:␣18">
    <img src="images/stop.jpg"></img>
    <font face="Comic␣San␣MS" size="4" color="#237381">
    Time:
    <font face="Comic␣San␣MS" size="2" color="#226351">
    <xsl:value-of select="Time"/></font>
    </font></p>
  </ul>
  <ul>
    <p align="left" style="margin-left:␣18">
    <img src="images/stop.jpg"></img>
    <font face="Comic␣San␣MS" size="4" color="#237381">
    Information:
    <font face="Comic␣San␣MS" size="2" color="#226351">
    <xsl:value-of select="Info"/></font>
    </font></p>
  </ul>
</xsl:for-each>
<p align="center">
<a href="javascript:window.close()">[CLOSE WINDOW]</a>
</p></body></html>
</xsl:template>
</xsl:stylesheet>
```

Listing C.12: Document Type Definition for Traffic Information

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT Info (#PCDATA)>
<!ELEMENT Record (TrafficRecord, Date?)>
<!ATTLIST Record id CDATA #IMPLIED>
<!ELEMENT Incident (#PCDATA)>
<!ELEMENT Time (#PCDATA)>
<!ELEMENT Traffic (Record+)>
<!ELEMENT TrafficRecord (Incident,Date, Time, Info)>
```