XML-Based Internet Performance Analysis

By

Nghi Lao

A Thesis
submitted to the Faculty of Graduate Studies
In Partial Fulfillment of the Requirements
For the Degree of

Master of Science

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba, 2000

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

XML – Based Internet Performance Analysis

BY

Nghi Lao

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements of the degree of Master of Science

I hereby declare that I am the sole author of this thesis.

I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I also authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Nghi Lao, 2000

The University of Manitoba requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

This thesis will define a method in which Web performance can be measured and summarize the results obtained from the application of this procedure to Extensible Markup Language or XML (can be described as a metalanguage that permits users to define their own markup language) performance analysis. It will also examine some common metrics used to measure the performance of the World Wide Web (WWW).

The procedure to measure performance includes the creations of similar HTML and XML applications that will be used as the base applications for testing. These applications will be a representation of the general content seen on the WWW today.

Tests were done with the server being located in Winnipeg, and the client located in Winnipeg or Calgary. Winnipeg and Calgary are two Canadian cities that are geographically separated by approximately 1300 kilometers. The tests in Calgary show the results when there is a greater network separation, while the Winnipeg tests are used to represent the opposite.

The test results show that XML applications that are highly stylized have better performance than comparable HTML application. The XML applications are able to serve 70%-100% more connections. Another interesting observation is the difference between the round trip times, an XML application overall has a round trip time that is 40%-50% less than the comparable HTML application.

For non-stylized applications created using XML, the performance is poorer than comparable HTML applications. Shown by the total average rounds served by the HTML application, which is 46%-66% more than the comparable XML application.

The round trip time of the XML application is also shown to be approximately 34%-44% longer then the comparable HTML application.

This thesis shows how XML and XSL style sheet can be used to improve the performance of highly stylized Web application, by using a separate XSL file to avoid repeating the styling tags.

Acknowledgements

I would like to thank TRLabs and MCI WorldCom for supporting and proposing the ideas for this thesis. Special thanks go to Jose Rueda for all his help, guidance, support and ideas throughout my stay at TRLabs. I also want to thank Professor Pawlak for letting me be one of his students. My appreciation and thanks goes out to all the help given to me by the TRLabs staff and students, for answering my questions; proof reading my papers and resetting my Web server a bunch of times. My two years spent at TRLabs was a great experience and I leave with many memories and friends.

The acknowledgments would not be complete if I did not mention my family, who help me immensely, thank you all.

Contents

List of Abbreviations and Acronyms

AIAG	Automotive Industry Action Group
ASP	Active Server Pages
CGI	Common Gateway Interface
CML	Chemical Markup Language
CSS	Cascading Style Sheets
DOM	Document Object Model
DTD	Document Type Definitions
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ICE	Information and Content Exchange
IETF	Internet Engineering Task Force
IIS	Internet Information Server
IPPM WG	Internet Protocol Performance Metrics Work Group
ISO	International Organization for Standardization
KOML	Koala Object Markup Language
MML	Mathematical Markup Language
NCSA	National Center for Super Computing Applications
NIMI	National Internet Measurement Infrastructure
PERL	Practical Extraction and Report Language
PHP	Personal Home Page
SAX	Simple API for XML
SGML	Standard Generalized Markup Language
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SQL	Structure Query Language
SSI	Server Side Includes
VBScript	Visual Basic Script
W3C	World Wide Web Consortium
WDDX	Web Distributed Data Exchange
WWW	World Wide Web
XIWT	Cross-Industry Working Team
XLink	XML Linking Language
XML	Extensible Markup Language
XML-RPC	XML Remote Procedural Calling
XMOP	XML Metadata Object Persistence
XPointer	XML Pointer Language
XSL	Extensible Stylesheet Language

List of Figures

List of Tables

Chapter 1

Introduction

The World Wide Web has been accepted as an effective method of communicating ideas and information. As the number of web users continues to increase on a daily basis. a need arises for better and more efficient methods for distribution of this information. Individual users, companies, and other organizations worldwide view the web as a medium to gather, distribute information, advertise, and market products. For companies, a web application has an audience of millions of potential clients. Thus creating the possibility of hundreds, if not thousands of requests to access their web site at any given time.

The problem is evident; there is a need for techniques to manage information on line. It is not only important to enable databases for online access, but it is also important to find a way to interact with the web browsers that users will utilize to access these online databases.

The Extensible Markup Language (XML) is a markup language used to describe and present data in a structured form. XML is a recommendation made by the World Wide Web Consortium (W3C) and it is a simplified form of Standard Generalized

Markup Language (SGML). In 1986, SGML became the international standard for defining descriptions of structure and content for different types of electronic document or the international standard metalanguage (a language that is used to describe another language) for markup, adopted by the International Organization for Standardization (ISO). XML is intended to make it easier for the implementation of SGML on the World Wide Web by leaving out the complex and less used parts of SGML. Therefore XML can be described as a metalanguage that permits the definition of custom markup languages. With the XML language specification being designed as an extensible data interchange format and a method for electronic publishing on the World Wide Web (WWW).

The HyperText Markup Language (HTML) is a predefined markup language and is a specific application of SGML that is used on the WWW. With HTML the tags are used to describe how data should be rendered by the computer, they're meant as a method for interactions between humans and computers. Therefore tags in HTML do not describe the data in an HTML document, but rather how the data should be displayed. Whereas in XML the tags are used to describe the data, thus giving meaning to an XML document. This makes XML documents understandable for humans while still retaining the ability for a computer to interpret and displaying the information.

HTML is the most widely used method to bring together text, images, sounds and videos to the WWW. The goal of XML is not to be a replacement for HTML, since they're both designed for different purposes. With XML being the web's language

for data interchange and HTML being the web's language for data rendering. XML is not intended to replace HTML but rather complement it. XML has the abilities to improve upon the many tasks that are currently being implemented by HTML.

XML is still a new technology in which to describe data in the electronic form. Currently the supporting languages that will assist in the deployment of XML on the WWW include: XML Linking Language (XLink) and Extensible Stylesheet Language (XSL). XLink and XSL are still working drafts and are not yet recommendations by W3C. XLink is a language that defines all the required elements in which to build links into XML documents. XSL is a formatting language that is used to transform an XML document into some arbitrary output structure (ex. HTML) which can then be displayed. By using Cascading Style Sheets (CSS) to present an XML document the content and presentation are separated, whereas with an HTML document content and presentation are combined. Just using CSS on an XML document without XSL can apply style to the document, but using XSL give more functionality in displaying a document than CSS on its own. With XLink and XSL, information defined using XML can be deployed on the WWW with more functionality than HTML. But XLink and XSL are still working drafts thus deeming them unstable languages, making them not yet deployable on the WWW.

Therefore SGML can be described as a method used to define thousands of different electronic document types; XML is an abbreviated version of SGML making it easier to define documents in electronic form for the WWW and HTML as just a language used to display documents for the WWW.

1.1 Thesis Statement

The thesis is concerned with determining the performance measures of XML. The general conception with XML is that it performs poorly compared with conventional content deliver methods (HTML). Performance problems encountered with XML are due to the fact that XML applications have larger file size and because of the tagging used to structure and describe the data. A larger application results in greater network traffic and server resources in delivering the content. It will look at how XML and its relating technologies can be used to reduce the application size as compared with conventional content delivery methods on the Internet. Along with how performance on the Internet can be measured and how a XML base Internet can benefit in terms of performance and functionality.

It also includes a look at the common metrics used in measuring Web performance and describes how these performance metrics are collected. An introduction to XML and Web technologies used to deliver XML will be discussed. The method to measure performance includes creating similar XML and HTML applications, from which tests will be performed to collect the performance data. Since the majority of the Web applications today are HTML files that are filled with text and images, our applications will represent how typical HTML Web applications seen today can be created using XML and show how XML may positively or negatively affect Web performance.

Chapter 2

Web Technology

2.1 Introduction

Content-delivery technology for the Internet is evolving dramatically. The technology used to deliver Web content today include instructions that are passed to the client for processing as well as those that are processed by the Web server. This chapter will give an introduction to some of the most popular Web techniques use to create and deliver the dynamic Web content. The discussion will generally be geared to open source development application since they are easily accessible and generally the most widely used, which includes discussion on server side and client side Web technology, along with an introduction to XML syntax and document creation.

2.2 Server Side

This section will give a brief discussion on the three top web servers used on the Internet today as determined by Netcraft [74]. It also includes a discussion on the most popular server side programming languages used to help deliver dynamic Web content.

2.2.1 Web Server

The three most popular Web servers used on the Internet today as declared by Netcraft [74] are Apache web server, Microsoft-IIS, and Netscape-Enterprise. As a result of polling over 9.5 million sites, from December of 1999, Netcraft determined the Apache web server to be first with a usage of 54.81%, followed by Microsoft-IIS at 24.26%, Netscape-Enterprise at 7.39% and other web servers making up the final percentage.

Microsoft Internet Information Server (IIS) is a built in Web server on Microsoft Windows NT Server operating system. Server side scripting on IIS is done mostly using ASP (Active Server Page) and the platform supported by IIS is Windows NT. Netscape-Enterprise Server is product of Netscape Communications Corp., it is a platform for which JAVA is used for development and provides multiple platform support.

The Apache Web server is a HTTP server that is developed by the non-profit Apache group and is based on the National Center for Super Computing Applications (NCSA) Web server. The Apache group along with developers via the Internet are the people who are currently building on the server and its modules. Although the Apache group has the final say on what will be included in Apache server. The Apache Web server is a fast, reliable and has multi platform supported, making it the most popular Web server used on the Internet as declared by Netcraft [74]. Since mid 1996 to the present Netcraft shows the Apache web server as the most widely used server on the Internet.

Testing by PC Magazine [72] shows Microsoft-IIS to be the fastest web servers to server static HTML pages and one of the top performers for dynamic web content (CGI tests) performers. Test results show Netscape-Enterprise Server to be a good static HTML performer with average dynamic performance. The Apache Server is shown to be an average static HTML performer and one of the top dynamic web content performers.

Both Microsoft and Netscape Web servers are commercial software, therefore a fee must be paid to use these products whereas the Apache Web server is free. All three Web servers have SQL database support along with support for XML. But support for XML is still in development and has not been totally defined yet. For example, with the Apache server XML parsers, XSL, and XSLT are still being developed and more information can be found at [23]. Netscape is developing ECXpert, an application that enables the exchange of commerce information between business systems that will provide an XML interface for sending and receiving XML documents.

2.2.2 Server Side Programming

As mention previously the Web server can pass web content to the client for processing as well as processing the content itself. The simplest forms of instructions that are executed on the server are known as server-side HTML or SSI (server side includes). A SSI page is an HTML page with embedded commands for the web server. With normal HTML pages the server does not parse the page but just sends it to the client. However with an SSI page, the server first parses the page and executes the instructions before they are sent to the client. The more complex methods of server

side processing include PERL (Practical Extraction and Report Language), JAVA, ASP (Active Server Pages), and PHP.

PERL is an interpreted language used mainly for text processing and on the Internet as a method to write Common Gateway Interface (CGI) scripts. When HTML forms are submitted, CGI scripts are used to process the information. CGI scripts are resource intensive because they require an additional process to be forked, involving the server to start a new process for each CGI script. Adrian Cockcroft shown in the March 1996 issue of SunWorld, that a 75 MHz uniprocessor SPARCstation 20 can handle about 20 requests per second when the server must fork a new process for each request. The same system can handle about 300 requests per second, if it does not have to fork a new process for each request. Therefore it would be better to use implementations that executing commands with an API running as a thread within the server's process. If CGI were required, a better alternative would be FastCGI, which acts like a server application and therefore eliminates the overhead of forking a new process. FastCGI is a proposed open standard implemented in the Apache Server that provides a better performance alternative for writing CGI in different programming language such as PERL, C, C++, and JAVA. Both Netscape and Microsoft have their alternative for CGI or application programming interface (API) called NSAPI and ISAPI, respectively. PERL also has support for many types of databases, a list can be found at [75] and there are also many PERL based XML parser available.

JAVA is an object-oriented programming language that is a creation of Sum Microsystems. When a JAVA program is running on the client side it is referred to as

applets and server side JAVA programs are referred to as servlets. JAVA has the ability to be platform independent, the JAVA Virtual Machine is what makes this possible. Therefore any machine can execute a JAVA program as long as it has the JAVA Virtual Machine. JAVA has a SQL database class and many JAVA based XML parser are also available.

ASP is a Microsoft implementation that allows server side scripting based on the Visual BASIC programming language. An ASP implementation only runs on the following servers, Microsoft Internet Information Server (IIS) and O'Reilly Website Pro. ASP has SQL database support along with XML support.

In 1994, Rasmus Lerdorf developed PHP(Personal Home Page) for personal use on his home page. As of November 1999, Netcraft's survey shows there are over 1 million severs using PHP. Currently PHP is only supported to run as a module for the Apache Web server. The PHP code is embedded into the HTML document and the code is executed on the server side, PHP syntax is similar to that of PERL and C++. PHP has support for talking to other services using protocols such as IMAP, SNMP, NNTP, POP3, and HTTP. A raw network socket can be opened and interactions can be done using other protocols. But PHP at its basic levels can accomplish what a CGI program can. What PHP is known for is as a method to access databases, because it does provide support for a wide range of databases. By using the Apache XML parser, along with some native PHP XML parser functions, it can provide support for XML applications.

2.3 Client Side

Client side execution of Web content is done using the Web browser, such as Netscape Navigator, Microsoft Internet Explorer, Opera, Mosaic, or Lynx. A Web browser works by creating a connection with the Web server, requests the data, then formats and displays the data on the clients machine. There are many different methods in which to achieve dynamic content on the client side. They include JavaScript, VBScript (Visual Basic Script), ActiveX and Java. XML support for Web browsers at the time of this writing is currently limited to Microsoft Internet Explorer (versions 4.0 and higher), but the next version of Netscape Navigator (version 6.0) will provide XML support.

JavaScript is an embedded scripting language, which is placed in an HTML document and is a creation of Netscape. JavaScript can also be used as server side scripting language but it is more widely used as a client side scripting language. All major browsers that are version 3.0 or higher support JavaScript on the client side. Although the more popular browsers support JavaScript, they do not implement JavaScript in the same manner. Therefore, certain browsers have there own additions of JavaScript and not all implementations of JavaScript will run on every browser. But what JavaScript provides is dynamic content that can be created on the client side with out having to access the server for data. With JavaScript, a Web page can react to what you're doing. Form elements can influence each other instantaneously and calculations can be made on the client side.

VBScript is a subset of the Visual Basic programming language, created by Mi-

crosoft to be used on the Web. What VBScript provides is a similar functionality to JavaScript, but is only supported by Microsoft's Internet Explorer browser.

ActiveX is a specification from Microsoft and is Microsoft's version of applets. ActiveX control can be written in any language as long as the client has the support for that language in which the ActiveX program was written. With ActiveX, the controls are downloaded and installed on the client's computer and are available to all client side applications. The downside to ActiveX are security issues and the fact that controls are executable files complied for the client's operating system, which require multiple executable to be created for the clients platform.

2.4 XML Syntax

This section will look at the markup syntax used in the creation of an XML document. It gives a description of the basic structure and rules that are required to create the XML document. Which includes an explanation on well-formed and valid XML documents, in addition to the syntax used in the creation of a DTD that accompanies valid XML documents. Also a general overview of some XML extensions will be given, these extensions give XML the functionality that is needed to be a suitable markup language used on the Web. The extensions that are explained include: Extensible Stylesheet Language (XSL), Document Object Model (DOM), Simple API for XML (SAX), Namespace and XML Linking Language (XLink and XPointer).

But first a description of the terminology that will be used in defining the elements used in markup. With the help of Fig. 2.1, the items that are part of a markup

element include: tags, attributes and element content, and are described as follows:

- Tags are the character strings (tag name) that are used to define the opening and closing part of an element.
- Attributes are the name and value pair that is contained within the opening tag of the element.
- Element is the entire character string, including the tag, attribute and the element content.

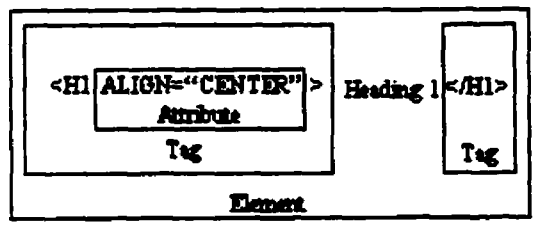

Figure 2.1: Typical markup element

From Fig. 2.1, a simple HTML markup element, the opening tag is <H1 ALIGN="CENTER"> with a tag name "H1" (first level heading element tag) and has an attribute named "ALIGN" with a value of "CENTER". The element content is the string value represented by "Heading 1" with a closing tag that is represented by </H1>. This is a representation of the content that is contained by a typical markup element and shows the parts and the terminology used in describing such an element.

2.4.1 XML documents

XML can be described as a metalanguage that permits users to define their own markup language to be primarily used on the WWW. Since XML is a metalanguage, documents created using XML do not have any predefined tags that authors must follow. This Allows for freedom in creation of documents that is not possible with the use of HTML. This gives the author the ability to create documents that have structure and allows for creation of data that is self-describing.

This section will explain the rules and syntax used in the creations of an XML document. It includes a discussion on the two types of XML documents: well-formed and valid XML documents. With the difference between well-formed and valid XML documents being, valid XML documents must follow the syntax outline set by a DTD or Schema (the grammar that defines the data structure and rules that must be followed when creating an XML document, set by the author). While well-formed XML documents must only follow the validity constraints set by the XML specification; these validity constrains must also be followed when creating valid XML documents.

Therefore well-formed documents are easier to create, since the author does not need to create an additional DTD document if it is not required. This Makes well-formed documents faster to process, since there is no need for additional processing to validate a document against a DTD or Schema. But a benefit with valid XML documents being, that all documents created with a given DTD or Schema will have the same syntax and structure. Which may not be the case for well-formed XML documents since there is no DTD or Schema to check the created document against.

2.4.2 Well-formed XML documents

XML documents that are described as being well formed must follow the specifications of well-formed documents defined by the XML specification [19] or follow the basic syntax rules of XML. A well-formed XML document generally contains two main sections, described as the prolog, and body.

2.4.2.1 Prolog

The prolog is the optional XML declaration that is found as the first element in the document. Although the prolog is an optional element it should be included in a document, since it contains useful information that can be used by the XML parser. The following is a XML prolog without any options and it must be defined in lower case letters:

<?xml version="1.0"?>

Next is a prolog with the two optional components, they are the character encoding being used by the XML document and if the document declaration is given by an external DTD.

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

The encoding attribute gives the encoding type used in the document; it helps other applications to determine the content of the document and is useful in mixed platform or mixed-language situation. If the encoding type is left out the assumed default type is usually set as UTF-8 or UTF-16 and is dependent on the parser used. The standalone attribute is used to tell that parser if there is an external DTD

required for the document. If standalone is set to "no" an external DTD is required, otherwise if the document does not requires an external DTD, if standalone is set as "yes" or is omitted completely, since "yes" is assumed as being the default value.

2.4.2.2 Body

The body of an XML document is where the data is contained, it must comprise of one or more elements and its what gives the document the tree like structure.

The markup used to define a XML element tag is defined with the less than character (<) and ending with the greater than character (>) or angle brackets, which enclose the tag name. The tag and attribute names used in the XML element are case sensitive, opening tag names must be the same as closing tag names; therefore a tag named "Car" is not the same as another tag name "car". Names must begin with a letter, underscore or colon and followed by letters, digits, hyphens, underscores, colons, or full stops. An exception to the naming construct is that names are not allowed to start with the "xml" string or any other string that matches these characters (for example Xml, XML, xML, etc.). The reason being that the "xml" string at the start of a name is reserved for standardization in the current and future versions of the XML specification.

Every XML element must have an opening tag and a closing tag, with the exception being the empty-element tag.

2.4.2.3 Opening Tag and Attributes

All opening tags must contain the tag name and can be followed by attributes, which are optional data and are not required. Attributes are additional data that is com-

prised of a name-value pair, where the attribute value must be enclosed by single quotes or double quotes, as shown by the unit and mpg, attribute name value pair below.

<fuelEconomy unit="mpg"> 24 </fuelEconomy>
<fuelEconomy unit='mpg'> 24 </fuelEconomy>

Attributes are a method in which additional information can be added to an element. For the XML element above, the tag name is "fuelEconomy", with an attribute named "unit" and a value of "mpg". By adding the unit attribute to this element the additional information of mpg (miles per gallon) is now known about the element content. As shown, the attribute value can be either enclosed in single quotes or double quotes and by adding attributes to an element provides the reader with more information about the element content.

There are two special attributes defined in the XML 1.0 recommendation they are xml:space and xml:lang. The attribute xml:space is used to preserve the text format or white space and is similar to the <pre> tag (preformatted text element) in HTML. The xml:lang attribute is used as a method to define the rendering of text or allows for creation of documents that are international. Since it allows definition of standard language codes [21], [22] or user defined codes. The xml:space and xml:lang attribute is applied to the element data and all other element that it encapsulates.

2.4.2.4 Closing Tag

The closing tag of an element is comprised of a forward slash (/), followed by the tag name and enclosed in angle brackets. The closing tag name must match that of the

opening tag name for the corresponding element.

2.4.2.5 Empty-Element Tag

The empty-element tag is used as a short hand to describe elements where there is no data, for example given the following data less element:

<price></price>

An equivalent empty-element tag would be:

<price/>

An empty-element tag contains a tag name, optional attributes that might be added, followed by the forward slash (/), enclosed in angle brackets. Another use for empty-element tags is a method to specify anchor points in a XML document, allowing for future programs to access these points in the document.

2.4.3 Structure of an XML Document

Now that we know how to create an XML element, this section will describe how these elements can be used to create an XML document such that it will be well-formed and structure will be given to the data. Rules of well-formed documents state that all XML document must contain a unique root element, have proper nesting of elements and entity references must be used in the place of reserved markup.

2.4.3.1 Unique Root Element

All XML documents must contain at least one element; therefore the one element that encloses all other elements in an XML document is referred to as the root element.

The root element must be unique and cannot be found anywhere else in the XML document. For example, a complete XML document of the following form in not well-formed:

```
<CAR>
    <MANUFACTURER> Honda </MANUFACTURER>
    <MODEL> Accord </MODEL>
    <CLASS> Midsize </CLASS>
</CAR>
<CAR>
    <MANUFACTURER> Honda </MANUFACTURER>
    <MODEL> Civic </MODEL>
    <CLASS> Subcompact </CLASS>
</CAR>
```

This is because it does not have a unique root element to make the document well-formed a root element would be required, as follows:

```
<CARLIST>
    <CAR>
        <MANUFACTURER> Honda </MANUFACTURER>
        <MODEL> Accord </MODEL>
        <CLASS> Midsize </CLASS>
    </CAR>
    <CAR>
        <MANUFACTURER> Honda </MANUFACTURER>
        <MODEL> Civic </MODEL>
        <CLASS> Subcompact </CLASS>
    </CAR>
</CARLIST>
```

2.4.3.2 Proper Nesting

Another rule that must be followed is elements must be properly nested. Nesting is where elements are embedding or constructed within another element. Nesting gives a parent/child relationship and is how XML document structure is created. For proper nesting to occur element tags must not overlap. No overlapping of tags means that the child element tag must be closed before the closing element tags of it parent/ancestors.

An example of improper nesting of elements is given by the following example, if we had a XML document written for some program that represents a for loop and an if statement within the for loop. The improper nesting is apparent in the code segment below, since the if statement in the for loop must be closed before we close the for loop.

```
<Program>
    <For i='1' to='10' increment='1'> ...
        <If i='5'> ...
    </For>
        </If>
</Program>
```

Therefore for proper nesting to occur the closing tag for the if statement </If>, would have to appear before that of the for loop </For>, the proper nesting of the code segment above is shown below, as follows:.

```
<Program>
    <For i='1' to='10' increment='1'> ...
        <If i='5'> ...
        </If>
```

```
    </For>
</Program>
```

This is what is meant by all child element tag must be closed before the closing element tags of it parent/ancestors.

2.4.3.3 Predefined Entity References

There is a set of reserved characters that are not to be used in the data source of the document or attribute values and must be replace by a particular character sequence referred to as the predefined entity references. These characters include the reserved characters for marking up an XML document and given by the following table .

Entity String	Usage
&amp;	Used to escape the & character (except within CDATA data section)
&lt;	Used to escape the < character (except within CDATA data section)
&gt;	Used to escape the > character (within a CDATA data section entity must be used if the > is followed by a]] string)
&apos;	Used to escape the ' character
&quot;	Used to escape the " character

Users can also define their own entity references, but these entity references must be defined prior to use in the DTD. A further explanation will follow on how entity references can be defined and is found in the DTD section to follow.

2.4.3.4 CDATA Section

The CDATA section is used as a technique to add text characters that would otherwise be interpreted as markup, without usage of entity strings. Since the XML parser does not parse the data that is contained in the CDATA section.

The CDATA section can occur anywhere document data can appear, but CDATA sections may not be nested. The syntax of the CDATA section is as follows:

<! [CDATA[...]]>

An example of how it can be used is when XML markup is required to be added to the document data as shown by the following section of an XML document:

```
<singleCar>
    <! [CDATA[
        <CAR>
            <MANUFACTURER> Honda </MANUFACTURER>
            <MODEL> Accord </MODEL>
            <CLASS> Midsize </CLASS>
        </CAR>
    ]]>
</singleCar>
```

2.4.3.5 Comments

Addition of comments into an XML document is added using the following syntax:

<!- - Comment Text - ->

The "Comment Text" can be any character string, with the exception of "- -" (double hyphen) which cannot be found in the comment text section. Plus the last character of the commented text cannot be a hyphen since this can be misinterpreted as part of the closing delimiter. Any entities found in the "Comment Text" section are not expanded and markup is not interpreted. Comments are not part of the data. If comments are placed within an element data they will not be interpreted as comments. In addition, placement of the comment must not appear within an element tag.

2.4.4 Valid XML documents

If an XML document is described as being valid it must obey and follow all grammar defined by the Document Type Definition (DTD) or schema. By creating a valid XML documents using a DTD or schema, this allows for different documents written by different authors to all have the same structure. All valid XML documents must follow the well-formed XML syntax and those defined by the DTD or Schema.

Therefore it can be said that all valid XML documents are well-formed, whereas well-formed XML documents are not valid XML documents unless they obey the rules set by a DTD or schema.

2.4.4.1 Document Type Definition (DTD)

The Document Type Definition (DTD) is the grammar that defines the data structure and is the rules that must be followed when creating an XML document. Schema on the other hand is an improved method in implementing a DTD, allowing for better data type definition and schema are created using the XML specification. The idea is to make Schemas easier to learn and more extensible than DTD when defining the documents structure. Schema are still in the working draft phase and are divided into two parts, structure [24], dealing with controls that describe the structural rules of a document and data types [25], dealing with definition of data types of the content.

The benefits of having a DTD is that the vocabulary used in the document is precisely defined, since all the rules of the vocabulary are contained in the DTD. In addition, by using a validating parser the XML document can be compared with its DTD to see if it follows the rules. This allows for different authors to create an XML

document all having the same syntax and structure.

2.4.4.2 DTD Declaration

The DTD can be declared in an external file or internally within the XML file. The benefit of having an external declaration is that the DTD can be reused by many different XML documents. An internal DTD allows for a single file to be sent that includes all the information, but if multiple documents are sent to a client requiring the same DTD, using an internal DTD makes for transmission of redundant data. Both an internal and external DTD can be used by an XML document, if declarations appear in both the internal and external DTD, the internal DTD declaration will have precedence. Otherwise by using an external DTD with additional internal DTD declarations allows for extra declarations to be added that are not present in the external DTD. This allows for fine-tuning of a predefined DTD to suit the authors requirements.

2.4.4.3 Internal DTD Declaration

The Internal DTD declaration is defined using the DOCTYPE tag name, followed by the root element name of the XML document. This is then preceded by the document declarations used to define the structure of the document, which is all enclosed in square brackets. The entire DTD declaration is then enclosed in the "<!" and ">". The following is a generic internal DTD declared within the XML document:

```
<!DOCTYPE rootElementName [ document declarations ... ]>
```

2.4.4.4 External DTD Declaration

As with the internal DTD declaration the external declaration is defined using the DOCTYPE tag name, followed by the root element name of the XML document. Instead of declaring the document declarations, the external DTD file location is defined as being PUBLIC or SYSTEM, as follows:

```
<!DOCTYPE rootElementName PUBLIC "public_identifier" "URL_of_DTD">
<!DOCTYPE rootElementName SYSTEM "URL_of_DTD">
```

The SYSTEM declaration is used to locate the DTD at the given URL. Whereas the public_identifier used in the PUBLIC declaration is a location string of the internal or external location of the DTD. If the DTD can no be found at location represented by the public_identifier string, the string representing URL_of_DTD is used instead as a URL naming the DTD file. The public_identifier consist of a text string that can be defined as being divided by double slashes using the following form:

"-//TRLabs//DTD carlist //EN"

The "-" character that begins the public_identifier is used to show that this is a non-registered identifier. If the identifier was registered with the W3C then a "+" character is required, and for an ISO standard a character string of "ISO" is required. The next section is an identifier for the author or organization, in this case TRLabs. Then followed by the keyword that is used to indicate the content format, in this case DTD, followed by a string used to indicate the document name, in this case "carlist". The final section is the language code, in this case "EN" used to represent English.

Therefore the PUBLIC identifier is generally used for well known DTD decla-

rations that are standardized or for authors that define a repository for their own DTD.

2.4.4.5 DTD Syntax

There are basically four markup declarations used in a DTD. They are element, attlist, entity and notation. These declarations are used in defining and constructing the DTD. The element and attlist declarations are used to define the XML elements and the attributes of an element, respectively. The entity declaration is used for declaration of reusable data and its primary design is to make XML creation easier. The notation declaration is used to declare data that is not XML and defines an external program associated with this data. An example is within a XML document a notation can be used to associate a JPEG binary data with a viewer to render the JPEG data. This section will be used to define the syntax of the four markup declarations and how they can be used.

2.4.4.6 Entities

By using entities, a predefined section of data can be referenced multiple times through a predefined name. Resulting in space being saved since larger repeated text can be replace by a small entity string and avoid retyping of repeated data. Entities can be referred to as being in one of the following predefined entities, general entity or parameter entity. Entities are also classed as parsed entity, where the replacement text will become part of the XML document or unparsed entity, where the replacement data is not XML or not even text therefore not required to be parse. As stated by the predefined entity reference section, there is a set of special entities

that are reserved for markup, and are referred to as predefined entities. Therefore all that needs to be defined is general entity and parameter entity.

2.4.4.7 General Entities

A user defined entity is referred to as general entities, allowing a name to be paired with a text string. This entity is declared by using the keyword ENTITY, a name and the text string that is associated with this name, shown as follows:

<!ENTITY projectTitle "XML-Based Internet: Performance Analysis">

Now that the entity is defined, to place the entity text into any element content, an ampersand is placed before the entity name followed by a semicolon. The following is how the projectTitle entity would be referred in a XML element:

<TITLE> Title: &projectTitle; </TITLE>

An external file can also be used to give the entity data and is given by the following form:

<!ENTITY myProjectTitle SYSTEM "http://nelson.win.trlabs.ca/projectTitle.txt">

Using the keyword ENTITY, a name and the keyword SYSTEM followed by the URL of the file declare this entity. The keyword SYSTEM can be replace with PUBLIC and a public identifier and fall back URL can be used as shown in the External DTD Declaration section.

2.4.4.8 Parameter Entities

Entities that are only used within the DTD are called parameter entities and allow changes to DTD constructs. Parameter entities are declared using the keyword ENTITY, a percent sign, a name and the text string that is associated with this name. The following is an example of a parameter entity:

<!ENTITY % unitAtt "unit CDATA #REQUIRED">

Now to use the parameter entity within the DTD the following syntax is used. A percent character is placed before the entity name followed by a semicolon, shown as follows:

%unitAtt;

A requirement of parameter entities is that they must be declared before they are referred. In addition the text that is added must be a valid declaration, if not the DTD will not be valid.

2.4.4.9 Element declaration

Element declarations in a DTD are used to define the syntax that elements must follow in a XML document. These element declarations are defined with the keyword ELEMENT and are followed by an element tag name. These names must follow the rules as those that were defined in the well-formed section previously. The name is than followed by the element content that can be one of the following four categories: empty, element, mixed and any enclosed in the "<!" and ">".

Element Content Category	Definition
empty	Elements with no data content and child elements contained within it, but it can contain attributes
element	This element contains child elements but not text data
mixed	This element is a mix of element and text data
any	Any elements content that does not violate XML sell-formed syntax

Therefore an example of how to create element declarations for the empty and any category above, is shown as follows:

<!ELEMENT car EMPTY>

<!ELEMENT carlist ANY>

The XML element that could be created from the empty declaration is <car/>. But by carlist as of type ANY the carlist element can have any combination of elements and text as long as the content between the tags is well-formed. This makes for placement of any well-formed XML document within the carlist element tags as being valid.

The "empty" and "any" element content category does not allow for definition of structure to be added to an XML document. By defining element content that fall in the "element" and "mix" content category a content model structure can be added to an XML document. Content model consists of some combination of element names, operators and the keyword #PCDATA, that is enclosed in parentheses. The #PCDATA keyword stands for parsed character data, which represents any text character but those that are used for markup. Any markup that is required must be replaced with an equivalent entity string.

The following table lists the content model operators that can be use in element definition to add structure to a XML document.

Operator	Definition
,	Separates items and shows the order they must appear
|	Separates items that list a choice of possibilities
?	Indicates items can appear one time or not at all
*	Indicates items can appear zero or more times
+	Indicates items can appear one or more times

For example given the following element declaration using the comma order operator:

```
<!ELEMENT car (manufacture, model, class )>
```

The following XML data that comes from the element declaration must have the elements appear in this order for it to be valid.

```
<car>
    <manufacture> ... </manufacture>
    <model> ... </model>
    <class> ... </class>
</car>
```

Next example will give an element declared using the pipe operator:

```
<!ELEMENT fuelEconomy ( city | highway )>
```

The XML data that comes from this element declaration can be of the following forms:

```
<fuelEconomy>
    <highway> ... </highway>
</fuelEconomy>
```

or

```
<fuelEconomy>
    <city> ... </city>
</fuelEconomy>
```

To add more complexity and create more complex structures requires the use of ?, * and + operators. They allow for the repetitions of elements and #PCDATA as shown by the following examples:

This element declaration is used to show how mixed content of different elements and #PCDATA can be combined in addition to the * operator. For mixed elements and #PCDATA must be separated by the | operator, and the #PCDATA type must be the first choice that appears.

```
<!ELEMENT testString ( #PCDATA | aaa | bbb | ccc )*>
```

The element declaration allows for and element testString to be a parent of zero or more character strings, or elements aaa, bbb, and ccc.

The next example will show how the ? operator can be used in a declaration of an element.

```
<!ELEMENT testString ( aaa?, bbb, ccc )>
```

With the element declaration above, the testString element can have zero or one aaa child element followed by a bbb and ccc child element.

The next element declaration is used to show how a + operator can be used in a declaration of an element.

<!ELEMENT testString (aaa | bbb | ccc)+>

With the element declaration above, the testString element can have zero or one or more child element aaa, bbb and ccc child element in any combination.

As shown above, a combination of these operators can be used to form a complex structure to represent almost any kind of data structure available.

2.4.4.10 Attribute declaration

The attribute declaration used by a DTD in defining attribute values for an element is defined using the following syntax.

<!ATTLIST elementName attributeName attributeType attributeDefaults>

All attribute declarations start with the keyword ATTLIST, followed by an element name that contains the attribute, the attribute name, an attribute type given by the attribute type table below or a character string, and an attribute default value. The following table gives a list of possible attribute default values available.

Attribute default values	Definition
#REQUIRED	The attribute must appear for the defined element.
#IMPLIED	The attribute may or may not appear for the defined element.
#FIXED fixedValue	The attribute may be declared; but it will always appear and is set as a default string value being represented by fixedValue.
Default value only	The attribute may appear if it does appear the default value is the assigned the value.

Since the #REQUIRED and #IMPLIED default values are easy to understand the following attribute example of a declaration using the attribute default value of #FIXED and default value only will be given.

<!ATTLIST fuelEconomy units CDATA #FIXED "mpg">

By using the #FIXED attribute default value when the <fuelEconomy> tag is declared the XML parser will include the unit="mpg" attribute to the <fuelEconomy> element, even if the element does not contain the unit attribute. The attribute value is fixed and therefore cannot be changed.

By just using a default value only, given by the following attribute declaration:

<!ATTLIST fuelEconomy units CDATA "mpg">

If the <fuelEconomy> element is declared with out any attributes the units attribute will be added with the default value of "mpg". But the units attribute can also be set to any character data string which is not possible by using the #FIXED attribute default value.

The following is a table of the possible attribute types that are available.

Attribute Type	Definition
CDATA	Character data strings only of any length that can't contain markup
ENTITY	Name referring to a external entity declared in the DTD
ENTITIES	Series of ENTITY names separated by white space
ID	Unique name within a given document
IDREF	Value referring to an ID declared to some element with the same value as the IDREF attribute
IDREFS	Series of IDREF names separated by white space
NMTOKEN	A name
NMTOKENS	Series of NMTOKEN names separated by white space
NOTATION	Take a name from a set of name indicating the notation types declared by the DTD, used to declare non XML data like GIFs or JPEGs
Enumerated	Excepts one from a set of user defined values

The enumerated data type is interesting since it lets the user defined a set of

items that the attribute value can be. An example of how this is declared is shown as follows:

<!ATTLIST price currency NOTATION (US | CANADIAN | BRITISH) "US" >

By using the notation attribute type the user can create a list of possibilities that the attribute value must be and assign values must be in the defined set. For the example above the price element has an attribute name currency that can have values of US, CANADIAN, or BRITISH. With the default value being US, by using a notation attribute type allows authors to set a predefined set of values to a given attribute.

Instead of covering the other possible attribute types and give an example for all the possible data types I would like to reference [4] that covers the topic and provides examples for each.

2.5 XML extensions

As mentioned previously, XML is a way to describe and structure data, therefore it is limited to what it can do. But by using the XML extensions or supporting languages additional functionality can be achieved. Below is a set of specifications of the extensions that support XML and a description of its value.

2.5.1 Extensible Stylesheet Language (XSL)

Extensible Stylesheet Language (XSL) is currently a working draft being developed by W3C [26], used to provide style and transform an XML document. The XSL language can be split into three separate languages: transformation (XSLT) [29],

accessing XML document structure (XPath) [30] and rendering/formatting (XSLF). Both XSLT and XPath are recommendations, while XSLF is still a working draft. The XSL design is based on two styling languages DSSSL (Document Style Semantics and Specification Language) and CSS (Cascading Style Sheets), both can be used to apply style to an XML document. DSSL is used as a language to style or transform a document, and is an ISO standard since 1996 [27]. CSS is a recommendation made by W3C [28], developed as an easy method in which to add style to an HTML and XML document. Where XSL differs from CSS is that it cannot be used with HTML, and it has the ability to transform a document. Difference with DSSSL is that, DSSSL is designed mainly for printed material and not only line documents. But what XSL provides is a styling and transformation language that is written using the XML syntax. There is also a submission for a proposed language called Spice, introduced by HP for a more CSS style sheet language for XML [31].

2.5.2 XML Linking

The proposed method in which linking is done in XML is called XLink or XML Linking Language (XLL) [36] and XPointer or XML Pointer Language (XPL) [37]. The XPointer specification is designed to allow for specifying single or multiple links within an XML document. Were as XLink, is designed to be used to link different XML documents and resources together. The links that are created using XLink can be categorized as being one of two types, simple links or extended links. Simple links have the same functionality as the HTML linking element, that is, links can connect in only one direction and has one resource identifier. With extended linking, multiple

resources can be connected, allowing for groups of resources to be linked from a single link. Extended links can also be used to filter the set of target links, allowing the user to perform real-time filtering of the linked resources.

Therefore, the problems with HTML linking that are solved with XLink and XPointer are:

- HTML links are inefficient use of bandwidth when only portions of a document are requested because the entire page is retrieved and displayed.
- HTML links can only return a single resource, therefore omitting related links that is also available. While with XLink a single link can relate to many resources, and is able to help searches return a list of related topics as well.
- HTML links have no knowledge of the structure of a document, and is dependent on the placement of anchor element. Therefore if the document data was changed the link could be invalid and rendered useless. Also with anchor elements placed in the document, requires them to be updated if the links are changed.

2.5.3 Document Object Model and SAX

The Document Object Model (DOM) [32] is a W3C specification that allows for a standardized method to access and manipulate document structure. The objects defined by the DOM allow for reads, searches, additions and deletions from a docu-

ment and defines a standard interface for accessing and manipulation of both HTML and XML data. To make the DOM platform neutral, W3C defines an interface for the different objects of the DOM but no specific method of implementation are given. This allows for the DOM to be written in any language and for legacy data to be accessed using the DOM. When the DOM is used to read the data, it parses the file and creates a tree like representation of the data in memory. This allows for faster access and manipulation of a document, but is not practical for large amount of data since it puts a strain on the system's memory. Currently there is a DOM level 2 specification [33] that is currently a candidate recommendation, which adds support for namespaces (allows for examination and modification), style sheets (includes object model, query and manipulation), filtering, event modeling and ranges (includes functions for manipulating large blocks of data).

There is another method to process the document structure similar to DOM, known as the Simple API for XML (SAX) [34]. The SAX is designed primarily using JAVA as an event-based interface, specifically a parser is used to read the document and report to the program about the symbols/events that it finds. This allows for larger data files to be parsed for events without loading the entire file into memory. Making it a simple and relatively fast way to get at the XML data, plus it is a good method to use when there is a large amount of data. The disadvantages of using SAX are that it has no random access to a document's data and complex searches are difficult to implement.

2.5.4 Namespace

Namespace is a method that ensures that element names are unique no matter where an element is used. In the W3C's Recommendations for Namespaces [35], a Namespace is defined as "...a collection of names, identified by a URI reference, which are used in XML documents as element types and attribute names.". The need for namespaces arises when there is collaboration between different XML documents that contain similar element names but have different meaning. If these elements from different XML documents were combined into a single document, the repeated element names would lose their meaning or would be undistinguishable. This problem of similar element name with different meaning can arise when XML is used on the Web or in large organizations. Therefore namespaces are required to distinguish between possible similar element names that might occur.

2.6 Summary

This chapter discussed the content-delivery technology used on the Internet, this included server side technology, client side technology, and XML syntax.

Server side web technology included a discussion on the web server along with a discussion on the programming languages that are used by the web sever. The client side discussion includes a brief introduction to the Web browser and programming languages that are used by them.

XML syntax was given along with the rules that must be followed in the creation of a XML document. This included an explanation of well-formed and valid XML

documents, along with the syntax of a DTD and how it can be used in valid XML documents to structure and validate the document.

Also a general overview of the extensions used by XML was given. These extensions give XML applications similar or more functionality than HTML documents, in addition to solving some problems faced with HTML. Which included the Extensible Stylesheet Language (XSL), that can give style to an XML document and can also be used to transform the XML document into another format. Linking is done using XPointer and XLink, with XPointer being used to specify single or multiple links within an XML document and XLink being designed to be used to link different XML documents and resources together. Document accessing and manipulation can also be done using Document Object Model (DOM) or Simple API for XML (SAX). With the DOM the entire document is loaded in memory therefore requiring more system resources, SAX is an event driven parser of the document that uses less resources and does not allow for random access or quicker access. In addition, namespaces was also explained as a method to avoid conflicts of similar tag names that might occur when XML documents are combined in the Web environment. Namespaces use a Universal Resource Identifier (URI), which allows for tags to be unique and distinguishable.

Chapter 3

Performance Measures

3.1 Introduction

The World Wide Web (WWW) can be described by the following three elements: servers, networks, and clients [3] [5] [12]. In measuring the performance of the WWW one must deal with collecting performance metrics for each of these three elements. Since there are many different clients that access the WWW by different configurations and connections, it would be hard to characterize the performance of the client. Therefore the majority of performance measures of the WWW deal with performance analysis on the server and network.

Server performance [9] [11] [12] is most often measured using the following four metrics: connections served per second, throughput (bytes/second), round trip time, and errors per second.

Metrics used to measure the performance of the Internet (networks) are currently still being standardized by the Internet standard organizations. The Internet Protocol Performance Metrics Work Group (IPPM WG) a work group for the Internet Engineering Task Force (IETF) is currently working on defining IP level metrics.

The work currently being done by IPPM WG [17] are still drafts or request for comments and outlines criteria in which acceptable metrics should follow. While the ITU Telecommunication Standardization Sector (ITU-T) has a new recommendation I.380 [18] and is an approved text but is not in its final form. The metrics proposed by both standards organization can be summarized as belonging to the following headings: connectivity, throughput, packet loss and packet delay.

The methods used to measure the performance of the WWW deals mainly with performing tests on the network and server elements of the WWW. This chapter will give an overview of the metric and methods currently used to measure the WWW performance. It describes the measures used to define the networks and servers performance as it relates to the WWW. It then gives an overview of methods and products that are used.

3.2 Network Performance Methods

This section includes discussion on standards that are currently implemented or being developed, plus some other work done by industry.

3.2.1 IETF

The IETF's Internet Protocol Performance Metrics Work Group (IPPM WG) is currently working on metrics that can be used to define IP performance. The IPPM WG [17] metrics include:

- connectivity
- packet loss (one-way packet loss)

- delay (one-way delay, instantaneous packet delay variation and round-trip delay)
- bulk throughput (TReno bulk transfer capacity and empirical bulk transfer capacity).

All the metrics are currently working drafts with the exception of connectivity, which is still a request for comment.

3.2.2 ITU-T

The ITU-T is currently the only standard organization that has recommended metrics to measure the performance of IP networks. ITU-T recommendation is I.380 [18] titled "Internet Protocol Data Communication Service - IP Packet Transfer and Availability Performance Parameters". It outlines the following parameters required to measure end-to-end or point-to-point IP performance:

- Population of interest (total set of packets being sent from source to destination)
- IP packet transfer delay
- IP packet error ratio
- IP packet loss ratio
- Spurious IP packet rate (packets not from predefined source, measured for a time period and then divided by the time period)
- Flow related parameters: IP packet throughput, octet based IP packet

throughput (parameters measured using probes, with parameter satisfying requirements outline in recommendation)

The I.380 recommendation also states methods and conditions that must be followed when measuring the metrics.

3.2.3 NIMI

The National Internet Measurement Infrastructure (NIMI) [1] [2] is another organization that is trying to defined methods and metrics used to measure the performance of the Internet. NIMI methods include using software tools, called probes to measure the metrics currently proposed by the IPPM WG, these probes are placed throughout the network to measure performance.

3.2.4 Surveyor

Surveyor [56] is a joint idea of Advanced Network & Services, Inc. (ANS), and the Common Solutions Group (made up of 23 universities). Surveyor uses several metrics including one-way delay and packet loss metrics proposed by IPPM WG. To obtain the metrics Surveyor uses measurement devices (Surveyor tools) that are deployed at each university and measures the performance of Internet paths between the sites in the study. The Surveyor tools are PCs (running FreeBSD) equipped with GPS antennae to provide time stamps accurate to ± 50 microseconds. Data is then stored into a database and put on the web so the participants in this experiment can access it.

3.2.5 XIWT

The article by Borthick [6] talks about metrics and techniques in measuring the performance of Internet service providers (ISP). That includes early methods introduced by Bellcore used in measuring the performance for the Automotive Industry Action Group (AIAG). This lead to a new method proposed by Cross-Industry Working Team (XIWT). XIWT is an organization whose goal is to "foster the understanding, development and application of technologies that cross industry boundaries" [16].

The metrics used by Bellcore and XIWT [16] to measure ISP performance include throughput, packet lost and round trip delay. XIWT states the minimal metric required to measure performance should include packet lost and round-trip delay. The metrics used by XIWT to measure reliability include reach ability (defined as a agent can send a packet to a test point and receive an acknowledgement from the test point), network service availability, duration of outage, and time between outage. Plus some ancillary metrics such as network resource utilization and DNS performance.

The procedure proposed by XIWT to measure these metrics includes placing measuring devices throughout the Internet to gather data simultaneously. This was proposed such that a standard procedure and metrics would be used to measure the performance of the ISP.

3.3 Servers Performance Methods

In this section we describe the metrics and methods that are used to measure the server performance of the WWW. The methods used to measure the performance of the servers deals with using software to obtain performance measures on the server. These measures are then used to specify the performance or could be used for input for a queueing network model of the server. Server performance [9] [11] [12] is most often measured using the following four metrics: connections served per second, throughput (bytes/second), round trip time and errors (errors/second).

The methods described in [12] describe how to use industry benchmarks to measure CPU, servers, and system level performance. With the aid of monitoring tools used to measure certain aspects of a client-server system and are located throughout the system. These measures are then used to obtain input parameters for representation of a queueing network of the system. A Performance analysis can then be done using the queueing network to determine maximum capacity or the bottlenecks in the system. Otherwise the measured data can be used to describe performance in general.

The procedure described in [5] deals with measuring workload by using logs generated from the server's activities. The logs are then used to characterize the workload of the server and how the workload affects the server's performance.

The procedure described in [9] [11] deals with measuring server performance through industry benchmarks. Tools that include WebStone by MindCraft [60], and Webperf a product of SPEC [61] (Standard Performance Evaluation Committee),

a non-profit organization that develops standard benchmarks and publishes official results.

The procedure described in [14] deals with using a queueing model to represent a single and multi web server system. Using a typical response time curve to represent the web server.

The procedure described in [15] deals with using actual solicited and unsolicited users around the world to test the performance of specified sites. Users were ask to record the time required to download files from the site and send the data back to the experimenters. A log of the server was used to measure the server side performance.

The test procedure used by [55] deals with measuring downloading times of specified HTML pages. The HTML pages were divided up into 4 categories: Large slow loading page (no special tags), short fast loading page (advanced html tags), frame page (specialty tags and Java applets) and VRML [71] (Virtual Reality Modeling Language - "an open standard for 3D multimedia and shared virtual worlds on the Internet."), and complex page (browser specific html tags). Users are then asked to download these pages and record the times; then send the results back to the experimenters.

The test procedure used by [3] deals with setting up a server using WebStone, a benchmarking tool that generates HTTP requests, and a monitoring tool called Webmonitor. This allows for experimenting multiple requests being sent to the server with all four server metrics mentioned previously being measured along with other system metrics like CPU and disk utilization

3.4 Measuring Solutions

This section deals with the software available to obtain the measurement for inputting into a queueing model or just used as a performance measure. Free measuring tools can be found at Cooperative Association for Internet Data Analysis (CAIDA) [58]. CAIDA provides a wide range of free software that can be used to measure and analyze network performance. Another method is to write script using the Simple Network Management Protocol (SNMP) tools [69], to obtain certain network performance measures. There are also monitoring and test tools that are usually provided with the web server that can also be used to obtain server performance measures. But the majority of the software mentioned below is not free, they tend to have limited time evaluation periods, or certain features not available unless the software is purchased. This section is broken into three subsections, describing network and server measuring tools. plus a section with tools that can be used to describe the network, server and client.

3.4.1 Network Testing and Measuring Tools

Cisco's Netsys Service-Level Management Suite [65] is used to monitor network routers and optimize performance of the network. Cisco's Netsys Service-Level Management Suite contains two modules: connectivity service manager and performance service manager. The connectivity service manager allows analyze of traffic flows, topologies, routing parameters, router configurations, and Cisco IOSTM software features. The performance service manager is used to collect routing configuration data

and can create service-level policies for connectivity, reliability, and security services. The performance service manager also can use VISTA (View, Isolate, Solve, Test, Apply) troubleshooting methodology to diagnosis and repair network problems. Netsys is a way to obtain important measures and provides solutions for many network problems.

HP Openview Tools with CiscoWorks200, designed Netmetrix [67] that uses the simple network management protocol (SNMP) to provide information about the networks performance.

INS Enterprise Pro is a network monitoring service provided by International Network Services [68]. INS will monitor a web site and then give web-based reports that include latency, throughput, errors and trends.

Real-time Applications Performance System (RAPS) by Resolute Software [59] used to monitor applications, servers and networks through agents. There are different agents to collect and measure the system performance and reports back to a central server at regular intervals.

SE Toolkit is a Unix performance monitor [64] designed by Adrian Cockcroft and Rich Pettit. Some measures that are collected by SE Toolkit include: TCP throughput, TCP connections, TCP retransmits, NIC rates, collisions, overruns and CPU plus disk usage levels. The SE Toolkit has analysis tools that can be used on the log files that are collected.

Visual UpTime by Visual Networks [70] is a WAN service level management system used for ATM, frame relay, leased line and IP/Internet services. Visual UpTime uses

monitoring agents and a database engine to measure and collect the performance of the network.

3.4.2 Sever Benchmarking Tools

WebStone by MindCraft [60], and SpecWeb96 by Standard Performance Evaluation Corporation [61] are used to send multiple HTTP GET requests to a web server. Therefore they are benchmarking tools used to measure the ability of a web server to handle HTTP GET requests.

WebBench [62] measures web server software performance by simulating clients requests to test for requests per second and throughput handled by the server. It allows test to be created by users for both static (HTML and GIF) and dynamic content (CGI, NSAPI and ISAPI).

3.4.3 Load Generators and Testers

The following is a list of tools used to test web performance by creating multiple client requests LoadRunner by Mercury [44], WebLoad by Radview [40], WebLoad by Platinum [47], WebART by OCLC Inc. [45], Socrates by Morph Technologies [46], Silk Performer by Segue Software [51], Load test tool by Portent [43], InetLoad by Microsoft [52], QALoad by Compuware [41], Forecast by Facilita Software [42], Performance Studio by Rational [49], and E-Test by RSW Software [50]. All these testers generate loads though a scripting language, or by recording an event and replaying it multiple times. These tools provide some measures of performance for all or a combination of client, network and server.

MS Web Application Stress tool, created by Microsoft's [53] is used to test a server's performance by reproducing multiple browser requests. It then gives the following measures get request per second, post requests per second, percent processor time, percent total processor time, and requests per second which then could be used to measure performance.

WebSizr by Technovations [48] is a load generator that can simulate up to 200 simultaneous HTTP users and record the results. By also using other supporting applications like DBSizr (test databases simulates SQL calls) and WebCorder (records HTTP transactions) along with WebSizr good simulations can be performed.

Keynote Perspective a service provided by Keynote [66] that uses over 100 cites across United States of America to download a specific page every 15 minutes. This will give webmasters information on how the users actually view their page throughout each cite.

VeloMeter [54] is a Java based HTTP server load tester that generates multiple client requests and measures the response times of the requests.

Baseline from TeamQuest [63] is used to measure performance of CPU, disk, buffer cache, memory usage, disk space utilization, network file system, TCP/IP, Sybase and Oracle database applications for Unix, NT and Unisys platforms.

Best/1 from BMC software, [57] runs on Unix and NT systems and collects data about the system resource usage, along with other measures.

3.5 Summary

The performance of the WWW can be described by measuring two elements, they are the servers and networks. The metrics used to describe the performance of the servers being connections served per second, throughput (bytes/second), round trip time and errors (errors/second). The metrics used to describe the performance of networks being connectivity, bulk throughput, packet loss, and delay.

The measures of performance can be obtained by software that creates multiple client requests and then measures the effects of the requests. These types of software are generally used to measure the capacity of the server and network. There are also monitoring tools that can be used to sample and test the performance which are geared toward maintaining performance.

This chapter provides information on the metrics and methods used to measure the performance of the WWW. Plus a list of tools that could be used to obtain measures used to test or monitor the performance of the WWW.

Chapter 4

Test Applications

4.1 Introduction

Web applications can be described as being static or dynamic. Static web applications are created once and the content does not change, this makes determining performance easier because larger pages require more server, network, and client resources. With dynamic web applications, the content that is delivered varies on the client's request, making performance more difficult to measure.

For a good performance comparison to be made, exact replications of XML and HTML Web applications were created. The test applications created for the test are dynamic since the data returned is dependent on the clients input. There are two different XML and HTML applications created to test the performance, and will be referred to as Application 1 and Application 2. Both applications contain a form for client input, from which a query will be made to return data in the form of text and graphics, with the theme of the applications being automobiles.

The form allows a client to enter the manufacturer's name, selected from a drop down list, the vehicle class and select a check box to display all data in the database.

For the HTML applications the form also includes a drop down list that allows for selection of the return data to be sorted and for the currency data element to be changed. An example of the form for the applications is shown in Fig. 4.1.

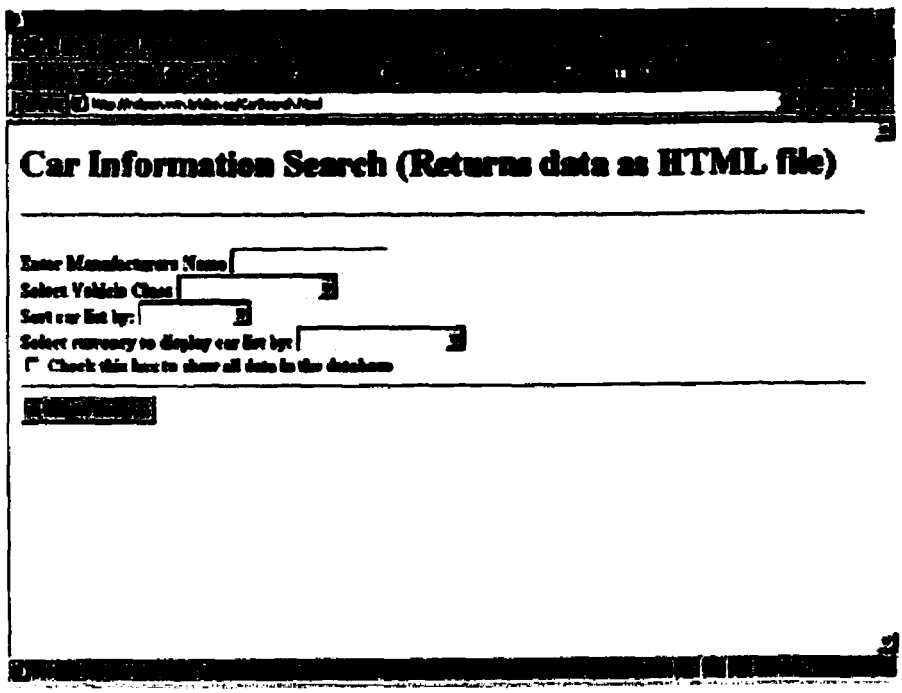

Figure 4.1: Input data form to query application.

Once the form is submitted, the Web server will process the request and sends the request to the MySQL database server. MySQL database server will than process the request and return the requested information back to the Web server.

MySQL is described as being a fast, multi-threaded, and multi-user SQL (Structured Query Language) relational database server. The phrase relational database can be broken down and described as follows:

- Database is described as being a collection of stored information that is structured and organized into tables. Each table is organized into rows

and columns; a row in a table represents a record. The records can have many entries, with each column in the table corresponding to one of the records entries.

- The word "relational" is used to state that this database is good at matching up information stored in one table to information stored in another table. This permits for information from different tables to be combined and allows for information to be gathered that cannot be generated from a single table alone.

To get the information out of the database requires the use of a language called SQL. The SQL language is the standard database language used to interact with the database and is used by all major database systems.

With the brief introduction to MySQL, we will now look at the tables that are used for our applications. Two tables were created for our database, named "car" and "engine". The record or row of the car table contains the following information: manufacturer, model, class, and image. While the record or row of the engine table contains the following information: model, liters, horsepower, rpm, city_mpg, highway_mpg, cylinder, and price. Since for a given car, there might be several different engines, the relational properties of MySQL allow us to match the model entries of the car table with the model entry in the engine table. The table setup and relationship between the two are shown by Figure 4.2.

By creating two tables and linking them by the model entry allows for data to be stored more efficiently. The data could have been stored in a single table, but for

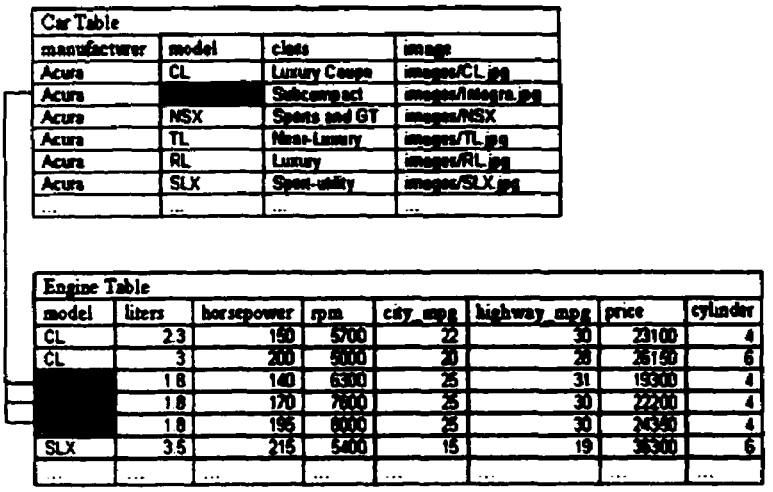

Figure 4.2: Car and engine table, linked by model.

each car with multiple engines, the manufacturer, class and image entries would be repeated if a single table was used. Therefore using two tables is more efficient and with MySQL being a relational database the data from the tables can be matched and linked.

The server side processing, including the MySQL database connectivity was done using PHP (Personal Home Page). PHP is a scripting language with syntax similar to that of PERL and C++, and is embedded into the XML and HTML documents. When an application is requested the embedded PHP code is executed by the server, and is used to generate the dynamic content sent to the client. PHP was built to run as a module for the Apache Web server, rather than as a CGI interpreter. By building PHP as an Apache module, every time a PHP script is interpreted the PHP module runs in the same address space or as part of the Apache process. This avoids the problems faced by spawning a different process, which affects performance negatively.

Therefore when the client submits the form, the Web server uses the PHP script to connect to the MySQL database and generate the query request. The database server will then return the results, from which PHP was used to generate the XML and HTML applications files sent to the client. Print statements in the PHP script were used to generate and tag the data returned from the database into either XML or HTML markup.

Although both applications are the same in appearance, the XML application is a little different in functionality. Within the XSL style sheet file for Application 1, scripting is included to give the client's side the ability to sort by manufacturer, model, and class, along with the ability to change the price data to a different currency value. The XML version of Application 2, allows for client side sorting of data, all that is required is a click of the mouse on the column value in the title row. This allows for sorting to be done on the table by manufacturer, model, class, engine type, horsepower, RPM, price, city MPG, or highway MPG in ascending order.

The sorting and data manipulation functions are not available on the client side once the data is received from the server with the HTML application. Instead, the client must enter these values into the form before the submission of the query or hit the back button on the browser and query the server again. Implementing this kind of functionality on the client side with the HTML application would be very difficult since data manipulation is difficult or not possible. Therefore the returned XML application from the query has a drop down list that allows for the data to be sorted and for the currency data element to be changed on the client side.

Sorting the received XML data on the client side is done using JavaScript and the Document Object Model (DOM). To sort the XML data, some Microsoft proprietary DOM extensions were used to interface with the XSL file. Currently the DOM level 1 specifications does not allow for stylesheet object modeling, querying and manipulation, but the DOM level 2 specifications, currently a candidate recommendation allows for this to be done. Therefore using Microsoft Internet Explorer to parse the XML and XSL files creates a document object model (DOM) for both files. Having the document object model for the XSL file allows access to the sort field object in the XSL file. The sorting object in the XSL file is found in the element that contains the "order-by" attribute. Using the DOM, the "order-by" attribute can be accessed from which the value can be changed and the way the data is sorted. The script to sort the data was created using a function that contains an input string with the desired sort value. The sort value is inputted from a form, with the submission of the form the sorting function would be called and the sorting would be done on the client side.

To change the price data values in the XML files requires the use of JavaScript and the Document Object Model (DOM). But the price value change does not require any special proprietary extensions and can be done with the methods provided in the DOM level 1 specifications. As with sorting of the data, a form was provided to allow the client to choose the currency value that would be displayed. When the client chooses a currency other than the default value and submits the form, the price change script is executed on the client side. The price data is found in the XML file

in the elements with a tag name PRICE. What is done in the price change script is the use of the getElementsByTagName("PRICE") method, which uses the XML file document object model to allow us access to all PRICE elements in the XML document. From which a loop was used to change the values in the PRICE element in our XML document to the new value.

The disadvantage of including scripting in the XSL style sheet to manipulate the data is that unnecessary data might be sent to the client, which increases the size of the style sheet sent to the client. The advantage is that the client will not have to make another trip to the server to manipulate data that is already on their machine. This will save on network traffic, plus reduce the load that is placed on the server.

The setup describes how dynamic content was created for both XML and HTML applications. The database setup used was kept the same for both XML and HTML applications along with application 1 and application 2.

4.2 Application 1

Application 1 is designed to be a nicely styled Web page, where data is returned in multiple table form along with an image pertaining to the data. The result from a client query contains images of the vehicle followed by a nicely formed table with the information of the vehicle under each image. The table contains a centered heading with the manufacturer's name, vehicle name, and the class of the vehicle; with a font size setting set equaled to four. This heading spans four columns and has a background that is orange in color. The heading is followed by four rows of data; the data is formatted in a table with borders, cell padding, and cell spacing

all set to zero. The first row listing the engine type of the vehicle, this data is centered with a white background. The second row represents the horsepower and RPM of the engine; the data spans two rows and has a light gray background. The third row is used to represent the fuel economy of the engine; the data spans two rows and has a white background. The forth row represents the price in American dollars with a background that is light gray in color. Depending on the vehicle being displayed, any additional engines available for that vehicle would be represented by additional columns in the table. The table ends with an additional row that is similar to the heading that spans four columns and has a background that is orange in color. Therefore a submitted query would return a list of vehicles, and each vehicle would have its image followed by a stylized table with data on that vehicle. An example of how application 1 is seen by the client is shown in Fig. 4.3.

Application 1 is intended to determine how XML/XSL can benefit from a highly stylized web application. With a highly stylized application there is repeated data that will benefit from a separate XML and XSL file. With XSL the styling data is only required once to style the entire page, and will not be repeated as is required by HTML. This may result in smaller web pages being generated and less network traffic for the XML application.

4.3 Application 2

Application 2 is a simple single table display containing text data only. All the resulting data returned is displayed in a single table with nine columns. The table contains a title row that is orange in color with the following column values: man-

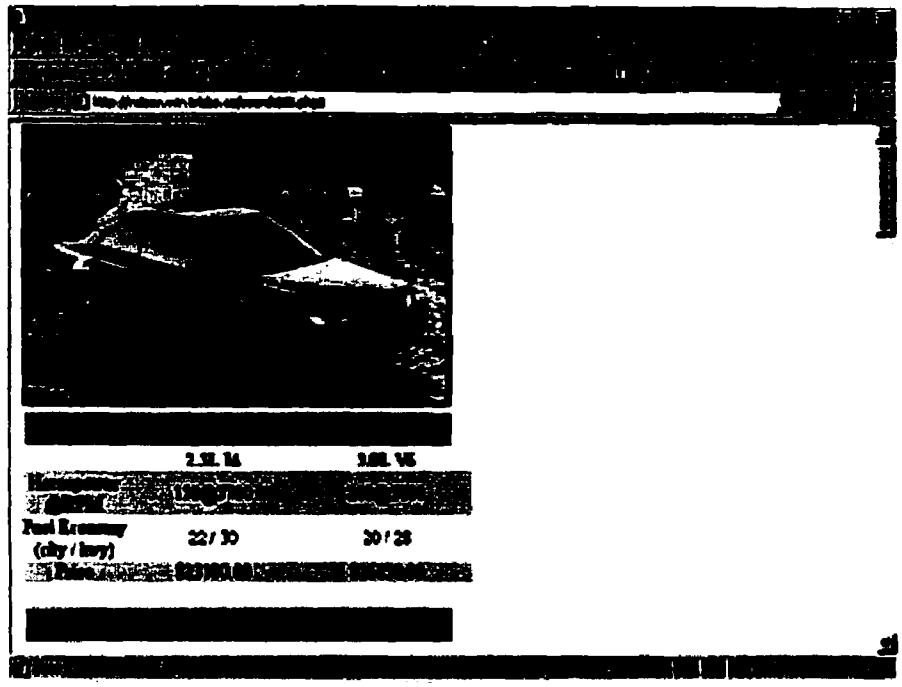

Figure 4.3: Application 1 - HTML web page (XML application displays the same data but has an additional form for data manipulation on the client side)

ufacture, model, class, engine type, horsepower, RPM, price, city MPG (miles per gallon) and highway MPG. The table is displayed with no borders and cell spacing, and the data is just centered in the appropriate column in the table. An example of how application 2 is seen by the client is shown in Fig. 4.4.

The idea behind Application 2 is to test how well XML might perform when data is not stylized heavily. This takes away the benefits of having a style sheet that will reduce the repeated styling data.

Application 2 also allows for testing to see if performance of the server will increase if the load placed on the server to sort the data is taken off. The sort option for the XML application is done on the client side using scripting, whereas with the HTML applications the database server does the sorting.

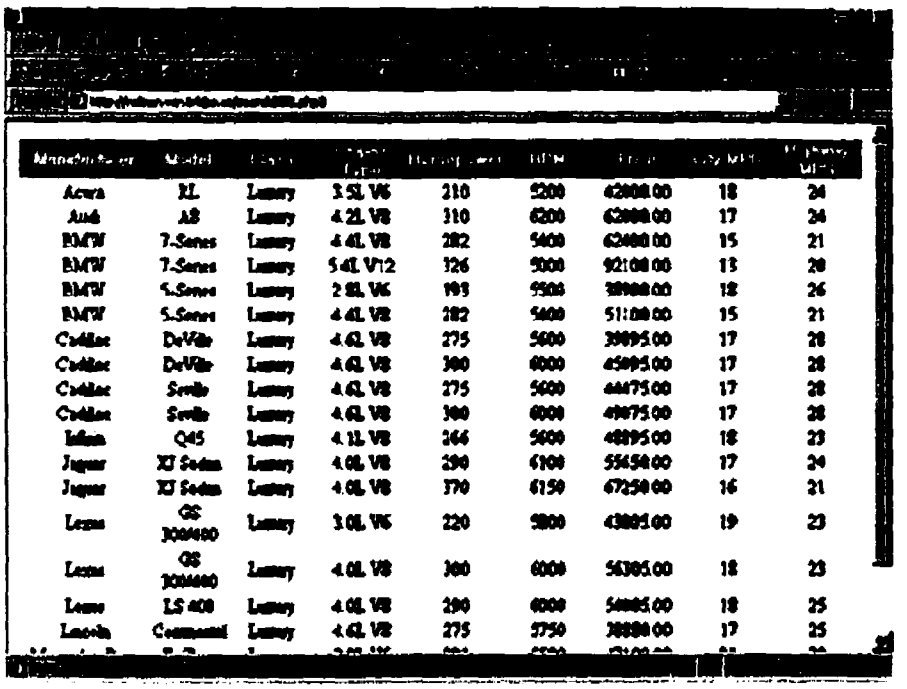

Figure 4.4: Application 2

Chapter 5

Test Setup

5.1 Introduction

This section will describe the setup used in the implementation of the XML and HTML applications, shown in Fig. 5.1, as well as the software and hardware used. The setup uses open source development applications since these applications are free, easily accessible and generally the most widely used on the Web. The following section contains a discussion on the server side setup, client side setup, and how the test was performed.

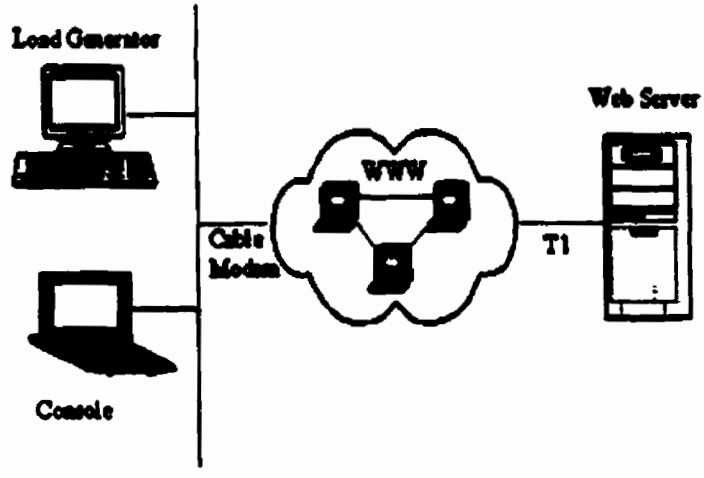

Figure 5.1: Test setup used to measure performance.

5.2 Server Side Setup

The server side setup consists of the Apache Web server (version 1.3.12) and MySQL database server (version 3.22.32), running on the Linux Mandrake (version 6.1) operating system. The Apache server is a HTTP Web server that is developed by the non-profit Apache group and is based on the National Center for Super Computing Applications (NCSA) Web server.

The server side scripting is done by PHP (version 4.0.0), since PHP provides native functions that support both XML and MySQL.

The hardware setup of the server was an Intel Celeron 466 with 64 MB of memory; and the server accesses the Internet through a T1 line.

5.3 Client Side Setup

The client was based on Microsoft Internet Explorer, because at the time of this writing it is the only browser that supports XML and XSL. Using Internet Explorer allows for testing of client side execution of XML applications with XSL.

5.4 Test Procedure

WebLoad by RadView Software Inc. [40] was used to load the server in addition to determining the performance metrics of the created XML and HTML applications. WebLoad can be described as a load-stress tester based on creating client requests. It provides a scripting language to create the actual client requests, these client request are called virtual clients and are used to emulate a Web browser. WebLoad provides

testing for information pages (pages that just contain data), interactive forms (pages which provide users interactions through forms), and search facility (pages which provide data entry to submit queries) allowing testing of the majority of web applications currently deployed on the WWW. This allows for simulations of multiple access to Web applications for real-time performance analysis. A trial version of WebLoad was used, that allows for 25 virtual clients to be simulated at the same time.

The WebLoad test session contains a console and load generator. The console can be described as the machine used to setup, run and controls each test session. With the console, the user can define the hosts that will participate, specify the program that will be executed, schedule, and view the performance results of the test. The load generator is the machine that runs the multiple simultaneous virtual clients request.

The load machine was setup on a PC that contains an AMD K6-3 400 MHz processor, with 128 MB. While the console is an IBM laptop with a Intel 233 MHz Pentium processor, with 96 MB. These machines are on a 10Mbps local area network, connected by a Linksys Etherfast Cable/DSL Router and access the Internet through a cable modem connection.

All test performed by WebLoad requires an Agenda, which is a file that is used to define the test to be performed. The Agenda for our tests consists of a request to the Web server for the form, from which the form field would be filled in and posted back to the Web server. To be realistic and fair, an input file was created that contains 50 valid input parameters of possible input that can be entered into the form, which would be used as input for the virtual clients in all of the test situations. Once the

end-of-file is reached, WebLoad loops back to the start of the file for input again, allowing for the 50 input values to be used an infinite number of times. Each test simulation was performed for a duration of 5 minutes. This testing procedure allows for different queries to be simulated simultaneously and is a fair representation that was maintained between tests, since all test queries are from the same set of 50 input parameters for both HTML and XML applications.

Tests were also performed with the sorting field left blank, designed as a test for general application performance. Selecting a non-blank value for the sorting field in the form allowed for testing of additional server side processing. This was done to see if the scripting included in the XSL file to sort the data on the client side was worth the additional bandwidth. A benefit of XML is the separation of data from the styling information, which allows for easy manipulation of the data on the client side.

Chapter 6

Results

6.1 Introduction

This chapter contains the collected performance results of the HTML and XML applications with the client located in two different locations.

When tests were performed on Application 1 the difference between the HTML and XML applications were not noticeable. This was because the images included with the application would be much larger than the entire data being transferred. For testing purposes the images were not included since the actual performance measurements resulting from the test of the HTML or XML application would be difficult to see.

Now an explanation of the performance metrics of total rounds and round trip time will be given, since it may be perceived as something other than what was measured. WebLoad [40] defines a round "as a complete execution of the agenda", and for our tests, the agenda represents the downloading of the form, filling plus posting of the form and the returned results in either XML or HTML format. Therefore the round trip time metric gives a measure in seconds it takes a client to download the form, post the data back to the server, and download the requested result, with the total

rounds being the total amount of time this process was successful.

The tests were taken with the server located in Winnipeg, and the client located in either Winnipeg or Calgary. Winnipeg and Calgary are two Canadian cities that are geographically separated by approximately 1300 kilometers.

The results of Tables 6.1-6.6 were taken with the Web server located in Winnipeg and load generator located in Calgary. Having the server and the load generator in different cities allows for greater network separation.

The results of Tables 6.7-6.12 are taken with the Web server and load generator located in Winnipeg. These results show how the applications would perform when both client and server are located in the same city, and the network separation is less of a factor.

The variance was also determined for the collected data, using the following formula:

$$Variance = \frac{n\sum x^2 - (\sum x)^2}{n(n-1)} \tag{6.1}$$

6.2 Calgary Results Applications 1

These results were collected on June 9th, 2000, with all tests times being indicated in Mountain Standard Time. Tables 6.1-6.3 are the performance results of the HTML and XML application 1, with the client located in Calgary and the server located in Winnipeg

Table 6.1: Server performance HTML - Application 1 (client in Calgary)

	Test 1 (2:40pm)	Test 2 (2:50pm)	Test 3 (3:10pm)	Test 4 (3:15pm)	Variance
Connections served per second	7.8	5.3	5.6	7.3	3.2
Total throughput (Bytes)	28507137	19221696	20353345	26465153	—
Throughput (Bytes/second)	95023.8	64072.3	67844.5	88217.2	—
Round Trip Time (Seconds)	2.502	3.796	3.425	2.463	0.805
Total Errors	0	1	0	1	
Total Rounds	2325	1565	1660	2157	—

Table 6.2: Server Performance HTML - Application 1 (with additional server side processing and client in Calgary)

	Test 1 (3:20pm)	Test 2 (3:30pm)	Test 3 (5:25pm)	Test 4 (5:30pm)	Variance
Connections served per second	8.7	8.2	7.6	5.9	2.3
Total throughput (Bytes)	31817544	29918558	27560711	21386700	—
Throughput (Bytes/second)	106058.5	99728.5	91869	71289	—
Round Trip Time (Seconds)	2.508	3.022	3.182	4.323	0.924
Total Errors	0	1	1	0	—
Total Rounds	2598	2443	2255	1748	—

For the results of Application 1, Table 6.1, 6.2 and 6.3, there is a noticeable difference in the amount of connections served between the HTML and XML applications. The XML application has an average connections served per second equaling to 13, compared with 6.5 and 7.6 as shown in Table 6.1 and Table 6.2 test of the HTML applications, respectively. This results in twice as many connections being served for the additionally loaded HTML application and almost that many for the non-additional loaded HTML application, resulting in more XML applications being served in the

Table 6.3: Server Performance XML - Application 1 (client in Calgary)

	Test 1 (2:55pm)	Test 2 (3pm)	Test 3 (3:40pm)	Test 4 (3:45pm)	Variance
Connections served per second	12.4	12.7	14.1	12.7	3.2
Total throughput (Bytes)	25609423	26243183	29143206	26234605	—
Throughput (Bytes/second)	85364.7	87477.3	97144	87448.7	—
Round Trip Time (Seconds)	1.754	1.738	1.515	1.542	0.073984
Total Errors	1	1	1	1	—
Total Rounds	3705	3796	4214	3793	

same time frame.

Even by serving more rounds, the XML application sent 3453096 bytes less or 3% less bytes of data, shown by the smaller throughput values.

Another interesting observation is the difference between the round trip times, the XML applications tend to have a round trip time that is 1.622 seconds or 50% less than the comparable additionally loaded HTML applications

The additional server side processing of sorting and changing content did not really affect the performance metrics negatively. The results of Table 6.2, with additional server side processing seemed to perform better than that of Table 6.1, without additional server side processing, this is an unexpected result. An explanation of why the non-additionally loaded HTML application performs worst than the regularly loaded HTML applications is explained by a more detailed look at the collected data. This includes the plot of the connection time, Fig. 6.1 and response time, Fig. 6.2, of test 1 of Table 6.2 and test 2 of Table 6.1, respectively, the extreme cases of the collected data for the HTML application.

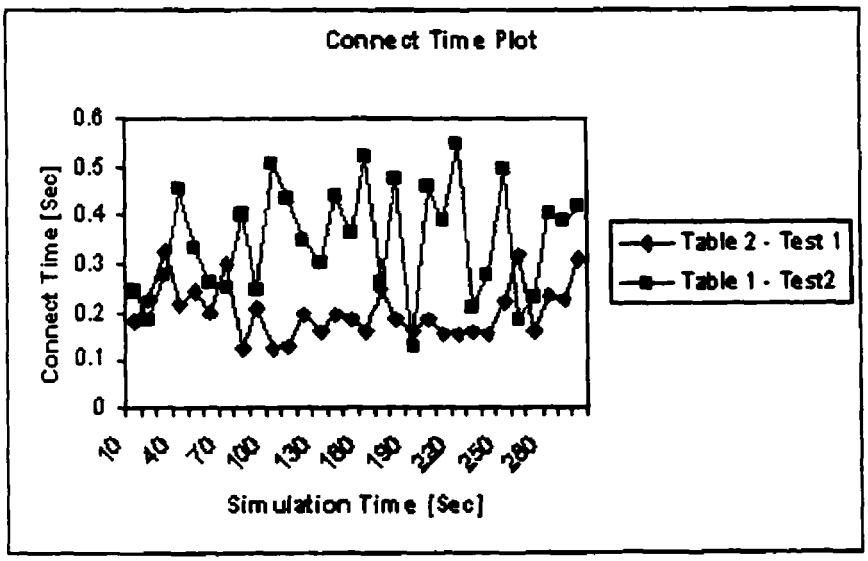

Figure 6.1: Connect time plot between Table 6.1- Test 2 and Table 6.2 - Test 1

WebLoad [40] defines connection time metric, as being "the time until a connection was achieved between the Client and the server (including the time it takes to establish the connection and receive the TCP/IP OK)". Response time metric is defined as, "the time required for the server to respond to a request sent by a client (starting from the end of the send including the time until the end of a blocked read of the incoming data)". Figure 6.1 shows that test run 1 from Table 6.1 has overall a higher connection time to the server, in addition to Fig. 6.2 that show a higher response time for test run 1 from Table 6.1. This observation was noticed between the test runs of application 1 in both Table 6.1 and Table 6.2. Figure 6.1 and Fig. 6.2 shows the noticeable difference in performance is caused by the poor network performance at the time the tests were performed, since the connection time and the response time should have been similar, which is not shown by Fig. 6.1 and Fig. 6.2.

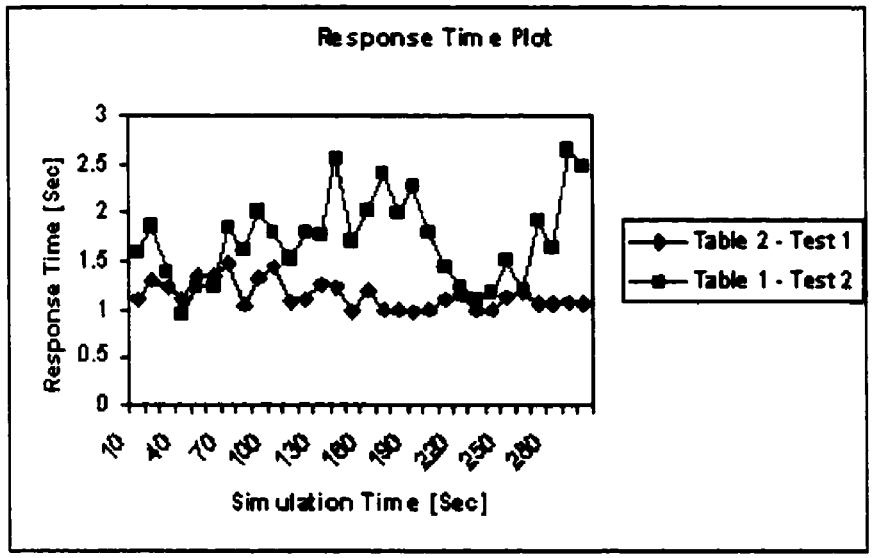

Figure 6.2: Response time plot between Table 6.1- Test 2 and Table 6.2 - Test 1

This shows that XML applications that have a separate style sheet, with repeated styling tags only required to be defined once for the data, results would be better performance than the comparable HTML applications. This may not be true in all cases since the larger data files will tend to show a larger difference in file size than those with less data, as discuss in the explanation of results section to follow.

6.3 Calgary Results Applications 2

These results were collected on June 9th, 2000, with all tests times being indicated in Mountain Standard Time. Tables 6.4-6.6 show the performance results of the HTML and XML Application 2, with the client located in Calgary and the server located in Winnipeg.

Table 6.4: Server Performance HTML - Application 2 (client in Calgary)

	Test 1 (4pm)	Test 2 (4:05pm)	Test 3 (4:55pm)	Test 4 (5pm)	Variance
Connections served per second	12.5	12.5	10.9	11.2	2.3
Total throughput (Bytes)	22203071	22213703	19430364	19912443	—
Throughput (Bytes/second)	74010.2	74045.7	64767.9	66374.8	—
Round Trip Time (Seconds)	1.702	1.861	2.150	2.167	0.096721
Total Errors	1	1	1	1	—
Total Rounds	3715	3715	3252	3333	—

Table 6.5: Server Performance HTML - Application 2 (with additional server side processing and client in Calgary)

	Test 1 (4:35pm)	Test 2 (4:45pm)	Test 3 (5:40pm)	Test 4 (5:55pm)	Variance
Connections served per second	10.8	11.3	9.9	11.6	2.6
Total throughput (Bytes)	19234781	20056173	17689131	20692007	—
Throughput (Bytes/second)	64115.9	66853.9	58963.8	68973.4	—
Round Trip Time (Seconds)	2.185	2.139	2.381	1.940	0.186
Total Errors	0	1	1	1	—
Total Rounds	3217	3357	2960	3462	—

For the results of Applications 2, Table 6.4, 6.5, and 6.6, the results are opposite to those obtained from Application 1. The round trip time of XML is larger than the HTML and the number of rounds served is larger for the HTML compared with the XML.

The average round trip time of the XML application is 3.003 seconds, compared with that of regular loaded HTML application at 1.97 seconds and the additional server side processing HTML application at 2.161 seconds. This shows the additional

Table 6.6: Server Performance XML - Application 2 (client in Calgary)

	Test 1 (4:15pm)	Test 2 (4:25pm)	Test 3 (5:10pm)	Test 4 (5:15pm)	Variance
Connections served per second	16.2	15.2	13.9	14.7	4.8
Total throughput (Bytes)	23232939	21753908	19818742	20976928	—
Throughput (Bytes/second)	77443	72513	66062.5	69923.1	—
Round Trip Time (Seconds)	2.749	3.130	3.353	2.780	0.229
Total Errors	1	1	1	1	—
Total Rounds	2418	2265	2062	2182	—

loaded HTML application requires one second or 34% less time to serve than the XML application.

The total average round served by the XML application is 2232, compared with that of the regular loaded HTML application at 3504 and the additional server side processing HTML application at 3249. This shows the HTML application can be served over a thousand more times or 46% more times than the comparable XML applications.

In the Application 2 test the additional loaded HTML application had a noticeable difference in the average amount of applications served compared with the non-additional loaded HTML application, this value being 255 additional rounds. There is also a smaller difference of 0.191 seconds in the average round trip time between the additionally loaded and regular loaded application. Plus the connections served per second by the additionally loaded application are 0.9 less than the regularly loaded or approximately an average of 1 connection less per second.

Application 2 shows that non-stylized applications created using XML does not benefit in terms of performance with a comparable HTML file, since these XML applications tend to be bigger and therefore suffer in terms of performance.

6.4 Winnipeg Results Applications 1

These results were collected on June 21st, 2000, with all tests times being indicated in Central Standard Time. Tables 6.7-6.9 are the performance results of the HTML and XML Application 1, with the client and server both located in Winnipeg.

Table 6.7: Server Performance HTML - Application 1 (client in Winnipeg)

	Test 1 (9:45pm)	Test 2 (9:50pm)	Test 3 (10:20pm)	Test 4 (10:25pm)	Variance
Connections served per second	13.5	12.8	12.1	11.6	3.2
Total throughput (Bytes)	49185689	46925931	44307849	42524710	
Throughput (Bytes/second)	163952.3	156419.8	147692.8	141749.0	—
Round Trip Time (Seconds)	1.819	1.450	1.369	1.423	0.089
Total Errors	0	0	0	1	—
Total Rounds	4015	3828	3611	3467	—

The results from Tables 6.7, 6.8, and 6.9 show similar results of those of Tables 6.1, 6.2, and 6.3, collected when the client was located in Calgary. They show that the XML application was able to perform better than the HTML applications.

Tables 6.7, 6.8, and 6.9 shows on average that the XML application was able to serve 2437 or 70% more rounds then the additionally loaded HTML applications.

Table 6.8: Table 8. Server Performance HTML - Application 1 (with additional server side processing and client in Winnipeg)

	Test 1 (9:55pm)	Test 2 (10pm)	Test 3 (10:30pm)	Test 4 (10:35pm)	Variance
Connections served per second	11.1	9.9	12.4	13.1	3.6
Total throughput (Bytes)	40468645	36039920	45293935	47983730	—
Throughput (Bytes/second)	134895.5	120133.1	150979.8	159945.8	—
Round Trip Time (Seconds)	1.947	2.321	2.052	1.867	0.143
Total Errors	0	1	0	1	—
Total Rounds	3305	2942	3699	3917	—

Table 6.9: Server Performance XML - Application 1 (client in Winnipeg)

	Test 1 (10:10pm)	Test 2 (10:15pm)	Test 3 (10:45pm)	Test 4 (10:50pm)	Variance
Connections served per second	19.2	18.5	20.6	20.8	4.4
Total throughput (Bytes)	39815058	38230902	42491369	42923371	
Throughput (Bytes/second)	132716.9	127436.3	141637.9	143077.9	—
Round Trip Time (Seconds)	1.252	1.372	1.116	1.177	0.025921
Total Errors	0	1	0	0	—
Total Rounds	5752	5526	6142	6204	—

Also on average the results show that the XML application had a total average throughput that was 1581382 bytes or 4% less then the additionally loaded HTML applications.

Another interesting result was the round trip time between the XML and additionally loaded HTML application. The average results show that the XML application takes 0.818 seconds or 40% less time to serve a round of the application.

The total errors that resulted when doing the test are minimal, therefore a final conclusion as to which application is better cannot be determine from these results. But these results tend to be in agreement with those collected in Calgary where XML performs better in all the measured performance metrics than the HTML applications.

From Tables 6.7 and 6.8 the results show that the non-loaded HTML application performs better than the loaded HTML application on average by 7.6% on the rounds served, 7.7% less bytes on the total throughput and has a round trip time that is 26% less. These results were expected since there was additional load placed on the server to sort and change the data, resulting in poorer performance. This was not observed in the test taken in Calgary, due to the poor network performance when the data was collected as shown above.

From these results, we can see that the XML application is able to outperform the comparable HTML application. Even though the XML application is served 70% more rounds, it is able to send 4% less total bytes of throughput and have a round trip time that is on average 40% less.

6.5 Winnipeg Results Applications 2

These results were collected on June 21st, 2000, with all tests times being indicated in Central Standard Time. Tables 10-12 show the performance results of the HTML/XML Application 2, with the client and server both located in Winnipeg.

Table 6.10: Server Performance HTML - Application 2 (client in Winnipeg)

	Test 1 (12:10am)	Test 2 (12:15am)	Test 3 (12:45am)	Test 4 (12:50am)	Variance
Connections served per second	26.5	27.2	27.2	27.0	0.81
Total throughput (Bytes)	47416428	48643946	48521590	48335966	—
Throughput (Bytes/second)	158054.8	162146.5	161738.6	161119.9	—
Round Trip Time (Seconds)	0.900	0.919	0.905	0.893	0.003
Total Errors	0	0	0	0	—
Total Rounds	7938	8143	8121	8090	—

Table 6.11: Server Performance HTML - Application 2 (with additional server side processing and client in Winnipeg)

	Test 1 (12:20am)	Test 2 (12:25am)	Test 3 (12:55am)	Test 4 (1am)	Variance
Connections served per second	25.0	27.2	26.9	27.0	1.4
Total throughput (Bytes)	44700489	48579400	48129434	48330485	—
Throughput (Bytes/second)	149001.6	161931.3	160431.4	161101.6	—
Round Trip Time (Seconds)	0.701	0.897	0.929	0.889	0.011
Total Errors	1	1	0	0	—
Total Rounds	7483	8131	8055	8089	—

The results from Tables 6.10, 6.11, and 6.12 show similar findings of those of Tables 6.4, 6.5, and 6.6, collected when the client was located in Calgary. They show on average that the additionally loaded HTML application was able to serve 3148 or 66% more rounds then the XML applications.

Also on average the results show that the additionally loaded HTML application had a total average throughput that is 863274 bytes or 3% more than the XML applications. This is due to the greater number of rounds of the applications that

Table 6.12: Server Performance XML - Application 2 (client in Winnipeg)

	Test 1 (12am)	Test 2 (12:05am)	Test 3 (12:35am)	Test 4 (12:40am)	Variance
Connections served per second	32.5	32.1	32.0	31.8	1.0
Total throughput (Bytes)	46446809	46066651	45763488	45423394	—
Throughput (Bytes/second)	154822.7	153555.5	152545.0	151411.3	—
Round Trip Time (Seconds)	1.541	1.497	1.563	1.454	0.005
Total Errors	0	0	1	0	—
Total Rounds	4844	4810	4773	4738	—

were served.

The round trip time of the additionally loaded HTML application, on average, is 0.660 seconds or 44% less time to serve a round of the application.

From these results we can see that the HTML application is able to outperform the comparable XML application. The HTML application is served 66% more rounds, but it sends 3% more total bytes of throughput and has a round trip time that is on average 44% less. Another interesting observation is the variance for connections served per second and round trip time (in seconds) for the data with the client located in Winnipeg is smaller than the data collected with the client in Calgary. This is expected since the data collected in Winnipeg has less hops than the data collected in Calgary. Since the duration of the test was only five minutes, the amount of data collected was not enough for a realistic variance calculation to be made for the performance metrics of total throughput (bytes), throughput (bytes/second), and total rounds. Therefore these variance values are not included since they do not represent a steady state value.

Chapter 7

Explanation of Results

7.1 Introduction

To clearly see why the XML version of Application 1 outperforms the HTML application we have to look at the file size of certain iterations of the application. Tables 13 and 14 are used to show the file size of Application 1 and the breakup of the file size by the number of elements returned (a single table of data or each vehicle represents an element). The differences between the HTML and XML files size are shown in Table 15, for Application 1.

To take a closer look at why the XML version of Application 2 does so poorly with respect to its HTML counterpart, we took a look at the file size of the comparable XML and HTML applications. Tables 16 and 17 show the file size of the HTML and XML version of Application 2 respectively, where an element in these tables is represented by a single row in the table of Application 2. While Table 18 represents the difference between the HTML and XML files size for Application 2.

7.2 Application 1

Tables 7.1 and 7.2 are a breakdown of the file size of the HTML and XML Application 1 respectively, for a single client request. These tables are intended to show the actual data that is requested from the server. Table 7.3 is then used to show the difference between the comparable HTML and XML application being transferred.

Table 7.1: Application 1 - HTML File Size

Number of Elements	Form (bytes)	HTML File (bytes)	Total (bytes)
1	1896	913	2809
5	1896	4532	6428
9	1896	8409	10305
12	1896	11138	13034
14	1896	12733	14629
18	1896	16469	18365
22	1896	21296	23192
70	1896	63185	65081
164	1896	154097	155993

Table 7.2: Application 1 - XML File Size

Number of Elements	Form +XSL File (bytes)	XML File (bytes)	Total (bytes)
1	7504	461	7965
5	7504	2252	9756
9	7504	4461	11965
12	7504	5859	13363
14	7504	6384	13888
18	7504	8288	15792
22	7504	12175	19679
70	7504	34608	42112
164	7504	82726	90230

From the difference shown in Table 7.3 the XML application is larger than the HTML application when the numbers of elements sent are small. But as the number

Table 7.3: Difference in HTML File Size as compared with XML Application 1

Number of Elements	Difference (bytes)
1	-5156
5	-3328
9	-1660
12	329
14	741
18	2573
22	3513
70	22969
164	65763

of elements increased, the data for the XML application is smaller than that of the HTML application. This is shown in Table 7.3, when the XML application reaches 12 elements, the served XML application is 329 bytes less than it HTML counterpart.

This is the explanation for why the XML application outperforms the HTML application. The reason for a smaller file size is the fact that with more elements being requested the formatting data for the XML file is constant. While for the HTML files the formatting data is dependent on the number of elements, as more elements are requested the larger the file becomes, due to the repeated formatting data. Even with the tagging that is required on the XML file that is not present in HTML, XML applications can be smaller than the comparable HTML application.

The smaller the files size the better the performance on the server side in addition to creating less network traffic, since less resources are needed in the creation and serving of the XML application. Another benefit of XML is that the XSL file is static and does not change therefore it can be cached by the server for faster access. Also better network performance is observed from the smaller round trip times that

are required to server the XML application. This shows that a highly stylized Web application can have better performance by using XML and XSL, as compared with HTML.

7.3 Application 2

Tables 7.4 and 7.5 are a breakdown of the file size of the HTML and XML Application 2 respectively, for a single client request. These tables are intended to show the actual data that is requested from the server. Table 7.6 is then used to show the difference between the comparable HTML and XML application being transferred.

Table 7.4: Application 2 - HTML File Size

Number of Elements	Form (bytes)	HTML File (bytes)	Total (bytes)
1	1732	877	2609
7	1732	1922	3654
15	1732	3350	5082
20	1732	4148	5880
27	1732	5549	7281
44	1732	8407	10139
71	1732	12474	14206
129	1732	21890	23622
290	1732	51627	53359

As shown by Table 7.6 the file size of the XML application is larger than the comparable HTML application for all cases. This is due to the tagging that is required in the creation of the XML application and the fact that a separate XSL file cannot benefit from repeated formatting tagging, as seen in Application 1. Therefore the XML application results in a larger file size for all cases.

Table 7.5: Application 2 - XML File Size

Number of Elements	Form +XSL File (bytes)	XML File (bytes)	Total (bytes)
1	4486	346	4832
7	4486	1757	6243
15	4486	3673	8159
20	4486	4776	9262
27	4486	6604	11090
44	4486	10499	14985
71	4486	16219	20705
129	4486	29173	33659
290	4486	68725	73211

Table 7.6: Difference in HTML File Size as compared with XML Application 2

Number of Elements	Difference (bytes)
1	-2223
7	-2589
15	-3077
20	-3382
27	-3809
44	-4846
71	-6499
129	-10037
290	-19852

7.4 Summary

This chapter shows the results of how XML and XSL were used to improve the performance of XML based applications. The performance benefit as a result of separation of the display tags from the data content. This generally results in smaller XML application file size for Application 1, the highly stylized Web content application, as compared with the HTML application. Due to the fact that XSL can be used to repeat all the display tags only once, while with HTML the display tags are required for each data element. But for non-highly stylized Web content, Application

2, the XML application performs poorly due to the fact that the created application is larger in size. This results from the fact that, there is not much formatting data that can benefit from a separate styling file.

By using XML, Web application can also be created with extra functionality that cannot be done using HTML. This functionality includes scripting sent to the client that can manipulate the data on the client's side. This can result in less network traffic, and reduced the load placed on the server.

Chapter 8

Discussion

8.1 Introduction

In this chapter we will look at the advantages that XML has over HTML, and the problems that can be solved by using XML. In addition to how XML can be used as a interchange technology between servers.

8.2 Why Use XML?

HTML is the most excepted markup language on the WWW, the question is why do we need a new markup language for the WWW? This question can be answered by stating the problems that are faced when using HTML on the WWW today. The problems with HTML are:

- Broken links - Since links in a web page are usually changing. Web masters are required to find these links on all HTML pages and change each link manually. This can cause required changes to be missed and broken links.
- Static tags - When using HTML developers can't define their own tags

to represent content. This leads to HTML extensions that are not standard or requires approval by W3C. Which leads to browser wars, where browsers having different tags. This can cause unreadable documents or require the creation of multiple documents to make them viewable by different browsers.

- Structure - HTML documents do not have any structure to describe the hierarchy and object representation of data in a document. Making searching time consuming because searching is limited to the full text, plus navigation and manipulation of documents difficult.

- Content description - HTML documents describe how the document should look and not what the document contains. This results in topic searches coming up with hits that are not related with the topic.

- Specialized/International characters - With HTML there is a lack of support for specialized and international characters. These are the characters required 2 or more bytes and those used by the science community for formulae.

- Reusability - HTML documents make it difficult to reuse the information. Since data for Web publishing, printed media, and data storage requires an HTML document be reformatted.

- Data interchange - With HTML data interchange is difficult because the data tags are used to describe how the data looks, making parsing data in an HTML document difficult. HTML is also an unreliable format to use

since there is no way to verify the received document.

The problem with HTML makes way for a new kind of markup language, this is where XML fits in. The introduction of XML is meant to add more functionality to the WWW, in addition to solving the problems faced by HTML today. Many think that XML is a replacement for HTML; this statement is only partially true. These two markup languages should be thought as being complementary, since they are designed for different purposes. With HTML being a method to display the data and XML used to describe and structure the data. Therefore XML should not be thought as a replacement for HTML, but XML can be thought as a way to solve the shortcomings of HTML and add more functionality to the WWW.

- Broken links - By using XML links can be defined through an aliases variable. This variable is then used to describe the link throughout the XML document. When the link changes the web master will only need to change the variable value and all links in the document will be change. similar to a constant variable in most programming languages.

- Static Tags - In XML tags are defined by the developer and are not required to be static. This allows an XML capable browser the ability to view the document even though the developers define their own tags. Allowing for browser independent documents to be created and creation of unique tags that better represent the documents without requiring the tags be approved by the W3C.

• Structure - All XML documents have structure defined by the developer. Structured documents allows for quicker searches because the entire documents does not have to be searched. Structure allows for easier creation of document maps making it faster to navigate the document. Plus manipulation of documents become easier, since the data now has a hierarchy and object representation making moving or changing of data in a page an easy task.

• Content description - With HTML, the tags are used to describe how data should be rendered by the computer, this is meant as a method for interactions between humans and computers. Therefore tags in HTML do not describe the data in an HTML document, but rather how the data should be displayed. Whereas in XML the tags are used to describe the data, thus giving meaning to an XML document. This makes XML documents understandable for computer to human interaction and computer to computer interaction, while still retaining the ability to interpret and displaying of the information. This allows for more accurate searches to be performed since the tags will describe the data and can be used to better the search.

• Specialized/International characters - With XML, data can be described using different encoding declarations. This allowing for XML to be used as a format for different encoding schemes to define character formats or languages. Some examples are Chemical Markup Language (CML) and the

Mathematical Markup Language (MML) written using XML for chemists and mathematicians to describe the complex characters and formulas.

- Reusability - With XML, data and display information are separate. When a user wants to display their information on a different medium they would apply a different style sheet to their XML document. With HTML, each medium that a user wants to display their information requires them to reformat or recreate the HTML document.

- Data interchange - With XML data interchange is a simple task, since all XML documents are self-describing, making them understandable by both computer to computer interaction and computer to human interaction. A Document Type Definition (DTD) can be used to check for syntax, structure, and validate an XML document. This makes it easier for developers to create applications to exchange XML document.

8.3 XML Application File Size

Using XML to structure and define data requires tagging that can increase the data's file size. But the added bytes used to tag the application are dependent on the document structure and tagging name used by the author. Therefore the file size of an XML file varies and is dependent on the author's implementation of the data. For example lets take a look at an example of a single car representation using the XML tagging format for Application 1, shown below.

```
<?xml version="1.0"?>
<CARLIST>
  <CAR>
    <MANUFACTURER> Jaguar </MANUFACTURER>
```

```
<MODEL> XJ Sedan</MODEL>
<CLASS> Luxury </CLASS>
<IMAGE> images/XJ.jpg </IMAGE>
<ENGINELIST ENGINES_AVAILABLE="2">
  <ENGINE>
    <TYPE> 4L V8 </TYPE>
    <HORSEPOWER> 290 </HORSEPOWER>
    <RPM> 6100 </RPM>
    <PRICE> 55650.00 </PRICE>
    <FEULECONOMY UNITS="mpg">
      <CITY> 17 </CITY>
      <HIGHWAY> 24 </HIGHWAY>
    </FEULECONOMY>
  </ENGINE>
  <ENGINE>
    <TYPE> 4L V8 </TYPE>
    <HORSEPOWER> 370 </HORSEPOWER>
    <RPM> 6150 </RPM>
    <PRICE> 67250.00 </PRICE>
    <FEULECONOMY UNITS="mpg">
      <CITY> 16 </CITY>
      <HIGHWAY> 21 </HIGHWAY>
    </FEULECONOMY>
  </ENGINE>
</ENGINELIST>
</CAR>
</CARLIST>
```

The XML tagging format for Application 1 shows how each data element is tagged. With additional cars being represented by additional <CAR> . . . </CAR> using the tagging scheme shown above. The creation using this tagging format is pretty generous in the tagging name and style used.

But the car data could have as easily been represented using more attribute representation rather than each data item having its own element, as shown as follows:

```
<?xml version="1.0"?>
<CARLIST>
  <CAR MANUFACTURER="Jaguar"
    MODEL="XJ Sedan"
    CLASS="Luxury"
    IMAGE="images/XJ.jpg">
    <ENGINELIST ENGINES_AVAILABLE="2">
      <ENGINE
        TYPE="4L V8"
```

```
            HORSEPOWER="290"
            RPM="6100"
            PRICE="55650.00">
            <FEULECONOMY
              UNITS="mpg"
              CITY="17"
              HIGHWAY="24">
            </FEULECONOMY>
          </ENGINE>
          <ENGINE
            TYPE="4L V8"
            HORSEPOWER="370"
            RPM="6150"
            PRICE="67250.00">
            <FEULECONOMY
              UNITS="mpg"
              CITY="16"
              HIGHWAY="21">
            </FEULECONOMY>
          </ENGINE>
        </ENGINELIST>
      </CAR>
    </CARLIST>
```

This representation shows the same data being displayed with the same structure and tagging names used to describe the data. But the data is now represented using attributes, by doing this, one can see that the XML file size is reduced, since most of the closing tags are eliminated. With this representation the functionality of the XML data is not affected because all that can be accomplished with the previous implementation is still possible with this implementation. The only thing that is affected is that the author has to create a different style sheet to represent this XML data.

Therefore XML file size is really relative to the author's implementation of the data, since different implementation can still have the same structure and functionality while having varying file size. But if applications are implemented using HTML the tagging format is fixed and there is generally only one way to tag data. Making HTML file size more predictable and performance easier to determine. While

XML application file size determination is difficult due to the fact that each authors implementation can be different, even if the application has the same appearance, functionality, structure and tagging name format.

8.4 Server Communication Using XML

The Web is observed as a client-server system, were a client would make a request to the server and the server returns information on the server or queries a database. But the web server cannot requests another web server for help in fulfilling the client's request. Using XML enables web applications to be created such that a web server can communicate with another web server for information or provide a service. Allowing for complex applications to be created from the current setup, with XML as the technology to provide a layer that can connect different systems. A list of techniques using XML for server-to-server communication is given bellow, which includes: XML-RPC, SOAP, Coin, WDDX, XMOP, WebBroker, ICE, and KOML.

For server to server communication over a network we need to first serialize the data, serialization is defined as "a process whereby a data term is structured in a simple one-dimensional format suitable for transport across the wire" [4] and its inverse, de-serialization. What serialization gives us is the ability to reduce platform dependencies and make the transmitted data less complex to manipulate.

Serialization of the data into XML gives us the ability to communicate simple or complex data types over the Internet. Since the transmitted data is just text and can be easily understood by all applications. The following is a list of methods in which data can be serialized into XML for Internet communication:

• XML-RPC (XML Remote Procedural Calling), allows serialization of data for Internet communication using HTTP. It is supported by many different implementations, where client/server side implementations include: Java, Perl, Tcl, ASP and PHP; client side only implementation include: Python, COM and AppleScript. A list of supported implementation along with specifications can be found at [76].

• SOAP (Simple Object Access Protocol) is similar to XML-RPC, but makes improvements on how the data being sent is tagged (based on using XML schema), allows for multi-reference of data (data defined once and reference multiple times) and greater control of HTTP header, allowing site administrators greater control of what can be done on the server. It also allows for multiple function calls be treated as a single transaction. This is not standard on XML-RPC but requires the user to implement this capability themselves. The Internet draft for SOAP can be found at [77].

• Coin is another serialization method that combines XML with Java. It's a way to improve on Javabeans, which uses Java serialization. By using Coins, it makes data less sensitive to changes and easier for exchange of data between applications. Coin allows linking therefore Coins can be returned to an XML document with an external reference, which can't be done using Javabeans, since it has no linking capabilities. By using XML-RPC the interface for communications is static, but Coin does not have

this limitation that allows for data elements to be added without updating all dependent programs. More information on Coin can be found at its homepage [78].

• WDDX (Web Distributed Data Exchange), it a method to communicate data structures between programming languages or a standard for language independent representations of XML data. Allaire develops the serialization/de-serialization modules, with supported implementation including: JavaScript 1.x, ColdFusion 4.0, COM, Perl, Java and PHP. WDDX is just a way to produce the data into XML object for communication and does not specify how the XML objects are transferred. It only requires that it be posted to a web page that can access the XML object. More information on WDDX can be found at its homepage [79].

• XMOP (XML Metadata Object Persistence) is a proposed way to allow for COM, Java and CORBA to inter-operate by serialization that does not tie it to a particular system object. With current intention to make XMOP complementary to XML-RPC or SOAP. XMOP uses SODL (Simple Object Definition Language) [80], an XML IDL DTD that allows objects to be created such that they are compatible with the IDL's used in COM and CORBA. More information on XMOP can be found at its homepage [81].

• WebBroker is a proposal method in which distributed object computing like COM and CORBA can be implemented on the Web. A submission

made by DataChannel to W3C describing WebBroker can be found at [83]. WebBroker is described as a method in which HTTP, XML and URI are used to define a software model such that it will not be hampered by incompatibles protocols when used over the Web, which is the case with COM and CORBA.

- ICE (Information and Content Exchange) Protocol defines a standard method to allow websites to exchange structured data using XML. ICE is being developed by the following companies: Adobe, CNET, Microsoft Corporation, National Semiconductor, News Internet Services, Sun Microsystems, Tribune Media Services, and Vignette Corporation. ICE allows for automatic exchanging and updating of data between Web sites without knowledge of the remote Web sites structure. ICE is still in development, its homepage is found at [84] and ICE version 1 notes can found at W3C notes at [85].

- KOML (Koala Object Markup Language) is a method in which Java is used to serialization/de-serialization Java Objects into an XML document. More information on WDDX can be found at its homepage [82].

The disadvantages of serialization/de-serialization using XML is the additional bandwidth and decrease speed. The additional bandwidth that is required is in the form of tagging the data being sent in the creation of an XML document. The slower speed involves the parsing and validating of the XML data received, before it can be passed onto the application. This requires the server to do more work on

each request that is received compared with the traditional communications methods COM/CORBA.

The traditional server to server communications using COM or CORBA are not really suited for the Internet, since it requires a degree of dependency and or platform related issues. This is where XML and the related techniques for serialization/de-serialization of data can be used to fill in the gaps or create new methods to allow servers to communicate over the Internet.

8.5 Summary

In this chapter we looked at why we need XML and some advantages that XML has over HTML. Also a discussion on server to server communications using XML was given, were traditional methods of COM or CORBA are not suited for the use on the Internet.

Chapter 9

Conclusion and Recommendations

9.1 Conclusion

A method was shown on how applications could be measured on the WWW using WebLoad. WebLoad is just one of many programs that are available in which testing of this type can be performed. The performance measures section found in this paper described numerous additional ideas and methods in which the WWW performance can be measured.

A general overview of content-delivery technology, along with a discussion on XML syntax and extensions were given. This included server and client side content creation technology along with how to use XML to create these applications. XML was also explained in how it can benefit content creation on the Web, which includes server to server communication and the advantages that is has over HTML.

The test results show that XML/XSL has a performance benefit that can be observed in applications that are highly stylized. This is because with XML the style that is applied to the data is separated from the data and is repeated once for all elements, compared to HTML where all data elements contain the style tags.

The tests show that using an XSL styling sheet to avoid repeating the styling tags, compensates for the tagging that is required in creating an XML document. The performance benefits of which include faster transfers, more rounds and less data being sent. In addition to having smaller files less server resources are used to create an XML application and since the XSL file is static it can be cached for faster access.

Another performance benefit that XML has is the ability to manipulate the data on the client side, which in most cases is not possible with HTML. This helps the client in avoiding another trip to the server for information that the client has already received. There was really no fair way in testing for this benefit but it is worth mentioning since the additional scripting does not affect the file size of the XML application that much. Even with the extra scripting, Application 1 file sizes are generally small and this helps the client in avoiding another trip to the server.

For data that is not heavily stylized, using XML/XSL does not benefit in terms of performance. For non-highly stylized application XML applications are larger, resulting in poor performance in terms of the server and network. This is the result of tagging that is needed to create an XML document that is not required with the comparable HTML applications.

With XML the tagging of data is done to describe and give the data structure, this tagging results in a larger file. But the results from this paper show that Web applications that are highly stylized can benefit from using XML/XSL to add more functionality, in addition to smaller applications file size when there is more data to represent.

9.2 Recommendations

Recommendations for future work on measuring XML performance includes collecting larger amount of performance data so that modeling can be done. The collected data should include a more detailed description of the server and network resources. This allows for more accurate input parameters for the model, which could be simulated using OPNET.

Bibliography

[1] A. Adams, J. Mahdavi, M. Mathis, and V. Paxson, "Creating a Scalable Architecture for Internet Measurement", http://www.psc.edu/networking/papers/nimi.html

[2] A. Adams, J. Mahdavi, M. Mathis, and V. Paxson, "An Architecture for Large-Scale Internet Measurement", IEEE Communications 36(8), pp 48-54, August 1998.

[3] J. M. Almeida, V. Almeida, and D. J. Yates, "Measuring the Behavior of a World-Wide Web Server", Seventh IFIP Conference on High Performance Networking (HPN), April, 1997

[4] R. Anderson, M. Birbeck, M. Kay, S. Livingstone, B. Loesgen, D. Martin, S. Mohr, N. Ozu, B. Peat, J. Pinnock, P. Stark, and K. Williams, Professional XML. Wrox Press, 2000.

[5] M. Arlitt and C. Williamson, "Internet Web Server: Workload Characterization and Performance Implications", IEEE/ACM Transactions on Networking Vol. 5. No. 5. pp 631-645, 1997.

[6] S. L. Borthick, "Why We Can't Compare ISP Performance-Yet", Business Communication Review, Vol. 28, Num. 9, pp. 35-40, Sept. 1998.

[7] D. Cintron, "Fast Track Web Programming: A programmer's Guide to Mastering Web Technologies", New York, NY: Wiley Computer Publishing, 1999.

[8] A. Homer, XML IE5. Acocks Green, Birmingham: Wrox Press, 1999.

[9] J. R. Lay, "Keeping the 400lb. Gorilla at Bay: Optimizing Web Performance". http://eunuch.ddg.com/LIS/CyberHornsS96/j.rubarth-lay/PAPER.html

[10] M. Leventhal, D. Lewis, and M. Fuchs, Designing XML Internet Applications. Upper Saddle River, NJ: Prentice Hall, 1998.

[11] R. E. McGrath, "Measuring the Performance of HTTP Daemons". NCSA, 1996. http://www.ncsa.uiuc.edu/InformationServers/Performance/Benchmarking/bench.html

[12] D. A. Menasce and V. A. F. Almeida, Capacity Planning for Web Performance Metrics, Models, and Methods. Upper Saddle River, NJ: Prentice Hall, 1998.

[13] W. J. Pardi, XML in Action. Redmond, Washington: Microsoft Press, 1999.

[14] L. Slothouber, A Model of Web Server Performance. http://louvx.biap.com/white-papers/performance/overview.html

[15] A. Smith, "Web performance metrics for online journals: monitoring and improving accessibility".
http://www.doe.gov/html/inforum98/apsmith.html

[16] Cross-Industry Working Team, "Customer View of Internet Service Performance: Measurement Methodology and Metrics", Sept. 1998.
http://www.xiwt.org/documents/documents.html

[17] "IP Performance Metric Draft and Request For Comments Page",
http://www.ietf.org/html.charters/ippm-charter.html

[18] "IPPM Documents Page", http://www.advanced.org/IPPM/docs.html

[19] "W3C XML Recommendation",
http://www.w3.org/TR/1998/REC-xml-19980210

[20] "Frequently Asked Questions about XML",
http://www.ucc.ie/xml/#FAQ-VALIDWF

[21] "ISO 639 language code page", http://sunsite.berkeley.edu/amher/iso_639.html

[22] "IETF's RFC 1766 language code ", http://www.ietf.org/rfc/rfc1766.txt

[23] "Apache Web Server homepage", http://xml.apache.org

[24] "Schema part 1: structures ", http://www.w3.org/TR/xmlschema-1/

[25] "Schema part 2: data types", http://www.w3.org/TR/xmlschema-2/

[26] "W3C XSL homepage", http://www.w3.org/Style/XSL/

[27] "DSSSL specification",
http://metalab.unc.edu/pub/sun-info/standards/dsssl/draft/

[28] "CSS homepage", http://www.w3.org/Style/CSS/

[29] "XSLT recommendation", http://www.w3.org/TR/xslt

[30] "XPath recommendation", http://www.w3.org/TR/xpath

[31] "Spice submission page",
http://www.w3.org/TR/1998/NOTE-spice-19980123.html

[32] "DOM level 1 recommendation", http://www.w3.org/TR/REC-DOM-Level-1/

[33] "DOM level 2 candidate recommendation",
http://www.w3.org/TR/DOM-Level-2/

[34] "SAX homepage", http://www.megginson.com/SAX/

[35] "Namespaces in XML",
http://www.w3.org/TR/1999/REC-xml-names-19990114/

[36] "XML Linking Language (XLink)", http://www.w3.org/TR/xlink/

[37] "XML Pointer Language (XPointer)", http://www.w3.org/TR/xptr

[38] "Web Performance Measuring Tools Article",
http://webreview.com/wr/pub/1999/01/15/feature/index3.html

[39] "Web testing resource page", http://www.softwareqatest.com/qatweb1.html

[40] "Web Load Generator and Analysis Software by Radview Inc.",
http://www.radview.com/

[41] "QALoad Web Load Generator and Analysis Software",
http://www.compuware.com/products/auto/

[42] "Forecast, Web Load Generator and Analysis Software",
http://www.facilita.co.uk/

[43] 'Portent, Web Load Generator and Analysis Software",
http://www.loadtesting.com/

[44] 'LoadRunner, Web Load Generator and Analysis Software",
http://www.merc-int.com/products/loadrunguide.html

[45] "WebART, Web Load Generator and Analysis Software",
http://www.oclc.org/webart/

[46] "Socrates, Web Load Generator and Analysis Software",
http://www.jump.net/ bpav/socrates/index.html

[47] "WebLoad by Platinum, Web Load Generator and Analysis Software",
http://www.platinum.com/products/

[48] "WebSizr, Web Load Generator and Analysis Software",
http://www.technovations.com/home.htm

[49] "Performance Studio, Web Load Generator and Analysis Software",
http://www.rational.com/products/index.jtmpl

[50] "e-TEST, Web Load Generator and Analysis Software",
http://www.rswsoftware.com/

[51] "Silk Performer, Web Load Generator and Analysis Software",
http://www.segue.com/

[52] "InetLoad, Web Load Generator and Analysis Software",
http://support.microsoft.com/support/DNA/Bundles/QA/loadtest.asp

[53] "Web Application Stress Test Tool", http://homer.rte.microsoft.com/

[54] "A Java-based load simulation and response measurement tool",
http://www.binevolve.com/velometer/index.vep

[55] "HTML Web Performance Test Page",
http://hjs.geol.uib.no/html/htmltest/t15506.htm

[56] "Surveyor Web Measurement Project", http://io.advanced.org/surveyor/

[57] "Unix and NT systems resource usage collector",
http://www.bmc.com/products/index.html

[58] "free software used to measure and analyze network performance", http://www.caida.org/

[59] "Real-time Applications Performance System (RAPS), tool to monitor applications, servers and networks", http://www.foglight.com/

[60] "WebStone, Web Server Benchmarking Tool", http://www.mindcraft.com/webstone/

[61] "SpecWeb96, Web Server Benchmarking Tool", http://www.spec.org/osg/web96/

[62] "WebBench, Web Server Benchmarking Tool", http://www1.zdnet.com/zdbop/webbench/webbench.html

[63] "Baseline, System resource measurement tool", http://www.teamquest.com/

[64] "SE Toolkit is a Unix performance monitor", http://www.sun.com/960601/columns/adrian/se2.5.html

[65] "Cisco Systems Homepage", http://www.cisco.com/

[66] "Keynote Internet performance measurement, diagnostic, and load testing services", http://www.keynote.com/

[67] "Netmetrix, network measurement tool", http://openview.hp.com/

[68] "INS Enterprise Pro is a network monitoring service", http://www.ins.com/

[69] "Simple Network Management Protocol (SNMP) tools", http://www.dart.com/powertcp/e/SNMPtrap.html

[70] "Visual UpTime, network measurement tool", http://www.visualnetworks.com/welcome.htm

[71] "Virtual Reality Modeling Language homepage", http://www.vrml.org/

[72] G. Alwang, "Web Servers", PC Magazine, May 5, 1998. http://www.zdnet.com/pcmag/features/webserver98/intro.html

[73] "Cold Fusion", http://www.allaire.com/products/coldfusion/index.cfm

[74] "Netcraft", http://www.netcraft.com/survey/

[75] "PERL Database Support", http://www.perl.com/reference/query.cgi?database+index

[76] "XML-RPC homepage", http://www.xmlrpc.com/

[77] "SOAP internet draft", http://msdn.microsoft.com/xml/general/soapspec-v1.asp

[78] "Coin homepage", http://www.jxml.com/coins/index.html

[79] "WDDX homepage", http://www.wddx.org/

[80] "SODL homepage", http://jabr.ne.mediaone.net/documents/sodl.htm

[81] "XMOP homepage", http://jabr.ne.mediaone.net/documents/xmop.htm

[82] "KOML homepage", http://www.inria.fr/koala/XML/serialization/

[83] "WebBroker Submission to W3C",
http://www.w3.org/TR/1998/NOTE-webbroker-19980511/

[84] "ICE homepage", http://www.idealliance.org/ice/

[85] "ICE W3C notes", http://www.w3.org/TR/NOTE-ice.html