Computer Science

Peter Johansson

A Prototype Application using XML:

With an introduction to XML

Bachelor's Project

2000:01

This report is submitted in partial fulfillment of the requirements for the Bachelor's degree in Computer Science. All material in this report which is not my own work has been identified and no material is included for which a degree has previously been conferred.

Peter Johansson

Approved, Date, 2000-01-10

Advisor: Stefan Lindskog

Examiner: Stefan Lindskog

Abstract

This thesis is written for Ericsson Telecom in Göteborg. The goal was to show how eXtensible Markup Language (XML) can be used, i.e., how to transform database content into XML and then present the content in another format, for example HTML. The application should have some connection to some Ericsson Telecom's products. An introduction to XML is given. Standards such as Document Type Definitions (DTDs), eXtensible Style Sheet Language (XSL), Cascading Style Sheets (CSS), eXtensible Query Language (XQL) and eXtensible Linking Language (XLL) are also presented. A prototype application that transforms database content into HTML using XML and XSL is described in chapter 6. Chapter 5 describes some of the tools used in the prototype application and chapters 2-4 give the basics about XML standards.

Contents

1 Introduction 1

2 eXtensible Markup Language (XML) 2

2.1 General Concepts 2

2.2 Tags 2

2.3 Prolog 3

2.4 Processing Instruction (PI) 3

2.5 White Space 4

2.6 !DOCTYPE Tag 4

2.7 Document Type Definition (DTD) 5

2.7.1 !ELEMENT Tag 5

2.7.2 !ATTLIST Tag 6

2.7.3 !ENTITY Tag 6

3 eXtensible Style sheet Language (XSL) 7

3.1 General Concepts 7

3.2 XSL Transformation Language (XSLT) 8

3.2.1 XSL Templates 9

3.3 eXtensible Query Language (XQL) 10

3.4 Cascading Style Sheet (CSS) 11

4 eXtensible Linking Language (XLL) 13

4.1 General Concepts 13

4.2 XML Linking Language (XLink) 13

4.2.1 Simple Links 13

4.2.2 Extended Links 13

4.3 XML Pointer Language (XPointer) 14

4.3.1 Absolute Addressing 15

4.3.2 Relative Addressing 15

5 Tools 16

5.1 General Concepts 16

5.2 Document Object Model (DOM) 16

5.3 SAX 16

5.4 XT 17
5.5 XP 17

6 Prototype Application 18

6.1 Overview 18
6.2 Detailed Description 19
6.2.1 Functions 19
6.2.2 Description of the XSL file 20
6.2.3 Description of the Transformation Tool 22

7 Conclusion 24

References 25

List of Figures

Figure 1. An empty tag 2
Figure 2. Another type of tagset with the same effect as figure 1. 3
Figure 3. Legal XML tags 3
Figure 4. The XML declaration 3
Figure 5. Valid PIs 4
Figure 6. Example with white space 4
Figure 7. A DOCTYPE declaration 4
Figure 8. Example of ELEMENT declaration 5
Figure 9. Example of a attribute list declaration 6
Figure 10. Example of an entity declaration 6
Figure 11. The use of XSL 8
Figure 12. Overview of relations in XML and XSL 8
Figure 13. CSS style flow. 11
Figure 14. Comparison between XSL and CSS 12
Figure 15. Links using locator 14
Figure 16. Links using group and document 14
Figure 17. First example of event based XML parsing. 17
Figure 18. Second example of event based XML parsing 17
Figure 19. Overview of the prototype application 18
Figure 20. Transforming XML into HTML 18
Figure 21. Example of an XML file 20
Figure 22. The script “xslt” for running XT 22
Figure 23. Command line arguments 22
Figure 24. Example of the index.html file 23
Figure 25. Example of a generated HTML page 23

1 Introduction

A time ago the eXtensible Markup Language (XML) seemed to be another typical software industry hype. Now it is here and it is growing all the time. More and more vendors start to implement XML into their applications. XML has a future in data manipulation and data transmission. It will be used to transfer structured information between different applications over the Internet. To explain the concept in short words we might say that XML is the way of representing data, XQL is the way of searching through the data and XSL is the way of transforming the information into another format.

2 eXtensible Markup Language (XML)

This chapter describes tags, white spaces, Processing Instructions, the doctype tag, DTDs and the specific tags inside DTDs.

2.1 General Concepts

XML is a subset of the Standard Generalised Markup Language (SGML) defined in ISO standard 8879:1986. It is designed to make it easy to exchange structured information over the Internet. The word extensible in XML comes from the fact that one has the ability to include any tags required. For example, if one have a database with many different columns, one might have a tag for each column, and that tag can be styled independently of the others.

2.2 Tags

Tags in XML is usually pairs like the following line of code: *<title>Hitchikers guide</title>*. Everything between the start and end tag is content. All tags in a document taken together is referred to as the "tagset". Because we have a basic XML rule that says, "all tags must be paired", i.e., text marked up with a given start tag must be paired with a given end tag. Another thing with tags is that all tag must be perfectly nested within one other. The code *<head><title>myTitle</head></title>* is illegal in XML because it is not nested properly, the tags overlap and that gives an XML document that is not well formed[1]. Tags in XML can also have an empty form, see figure 1 and 2 to see two similar tagsets that have the same function. Tags in XML are case sensitive, i.e., *<img><ImG><IMG>* is different. Figure 3 shows the different variants of tag names that can be implied. The rules are that they must begin with an underscore or a letter. Subsequent characters in the name may include letters, digits, hyphens and periods. However they may not include white space [3].

<network name="AXE"/>

Figure 1. An empty tag

[1] Well-formed means that XML documents complies with XML rules such as proper tag nesting.

<**network name=”AXE”> </item>**

Figure 2. Another type of tagset with the same effect as figure 1.

<AXE>
<Phone>
<network>
<cellular1>
<wireless.markup.language>
<P.J>
<_3wap>

Figure 3. Legal XML tags

2.3 Prolog

When structuring an XML document the documents beginning usually is called the prolog. The prolog usually contains two optional parts. First, the XML declaration that identifies the version of the XML specification to which the document conforms see figure 4. One should always include one even if it is optional just to specify the version of XML in use. This should be done so one can identify old documents in the future when using new versions. The second part in the prolog is the document type declaration which indicates where the grammar rules or document type definition (DTD) are placed [6], see chapter 2.7 for an explanation. It may include the XML declaration itself, the Document Type Declaration (see chapter 2.6), comments and PIs (see chapter 2.4).

<?xml version=’1.0’?>

Figure 4. The XML declaration

2.4 Processing Instruction (PI)

Processing Instructions (PIs), are for occasions when you need to say something to a computer program without a reference to the DTD and without changing the way that other computer programs process the document. PIs are intended to be software specific markup. It might be a sound file, movie clip, some picture or just a help to format your document in an extra “spicy” way, see figure 5. An example would be to connect to a CGI program via a PI so for example the current date and time could be obtained [1]. In figure 5, the mp3walkman

is called the target of the PI, and version, frequency and bitrate is the properties of the PI, gcc and acrobat are other examples of valid PIs [3].

<?mp3walkman version=”1.0” frequency=”4.5kHz” bitrate=”160kbps”?>

<?gcc Helloworld.c?>

<?acrobat document=”whitepaper.pdf”?>

Figure 5. Valid PIs

2.5 White Space

XML defines white space to be any sequence of one or more space, tab, carriage return, or line feed characters. White spaces can be used inside markup tags to make the text more readable. For example, a poem might lose some of it meaning if it is not written in a specific way with big indents on certain parts, see Figure 6 [2].

<poem>

Oh what a tangled Web we’ll weave,

… If our ‘ML’s to succeed

Far beyond our parser’s sigh

Let us validate at first try

</poem>

Figure 6. Example with white space

2.6 !DOCTYPE Tag

A confusing part in XML is that, at the first glance, document type declaration is indistinguishable from document type definition (DTD). What makes this even more confusing is the fact that the two terms are related. The doctype declaration tells the processing software which DTD to use to understand the document. The syntax for a document type declaration can be viewed in figure 7.

<!DOCTYPE nameinfo

SYSTEM “http://www.name.org/name.dtd”>

Figure 7. A DOCTYPE declaration

The “nameinfo” is in general just a label for the document type. Specifically it identifies the tag that will be used as the root element in the XML document, which makes it easier for the parser to determine where the prolog ends and the root element begins and ends. The keyword SYSTEM, followed by a Uniform Resource Identifier (URI) indicates where the DTD is located [2]. URIs are in contrast to Uniform Resource Locators (URLs) a general architecture for locating resources on the Internet, that focus a more on the resource and a less on the location.

2.7 Document Type Definition (DTD)

The Document Type Definition (DTD) is not strictly necessary in XML documents, but a complete DTD makes sure that the document is valid[2]. The DTD should be included so the XML processor can check weather the XML document obeys the rules you have set-up [4]. It is possible to have the DTD inside your XML document as well as an external DTD. It is, however, a good idea to separate the DTD from the document so you just have the XML tags in your XML document and a DTD for all your XML documents. The structure of the DTD include [7]:

- Definitions of the tags that are used to mark up an application’s document.
- A description of the allowed structural relationships between tags, for example, a <*page*> may appear inside <*book*> but not the other way around.
- Specification of the sequence in which tags must appear <*preface*> must come before <*chapter*> for instance.

2.7.1 !ELEMENT Tag

The element declaration identifies the names of elements and the type of their content. An example of an element declaration is [6]:

<!ELEMENT EMAIL(TO+, FROM, CC*, SUBJECT?, BODY?)>

Figure 8. Example of ELEMENT declaration

The declaration in figure 8 indicates that:

- The “TO” element may appear one or more times.

[2] An XML document is valid if its structure and element content is formally declared in a document type definition (DTD).

- The “FROM” element must appear exactly once.
- The “CC” element may appear zero or more times.
- The “SUBJECT” element can be included zero or one time.
- The “BODY” element may appear zero or one time.

The commas indicate that the elements must occur in succession [13].

2.7.2 !ATTLIST Tag

The attribute declaration defines the structure of an element and the kind of contents it have [6]. In other words, the attribute provides additional information about the element or the content of that element. Each attribute in a declaration has three parts: a name, a type and a default value. The name entry is the name of the attribute. The type says weather it is a text string, if it is PCDATA, CDATA, etc [13]. In Figure 9 TO is the name. CDATA is the type. #REQUIRED means that TO and FROM must be filled in. #IMPLIED means that CC, SUBJECT and BODY can be left out [3].

```
<!ELEMENT EMAIL(TO+, FROM, CC*, SUBJECT?, BODY?)>
<!ATTLIST EMAIL TO          CDATA   #REQUIRED
<!ATTLIST EMAIL FROM        CDATA   #REQUIRED
<!ATTLIST EMAIL CC          CDATA   #IMPLIED
<!ATTLIST EMAIL SUBJECT     CDATA   #IMPLIED
<!ATTLIST EMAIL BODY        CDATA   #IMPLIED >
```

Figure 9. Example of a attribute list declaration

2.7.3 !ENTITY Tag

The ENTITY tag makes it possible to declare commonly used text within the DTD as a text entity. An example of a typical test entity, see figure 10.

```
<! ENTITY kau "Karlstad University" >
```

Figure 10. Example of an entity declaration

Once the declaration is made inside a DTD, one may use that entity reference on the form &kau; in place of the full name of the university. The great advantage with this is that if the name of the university is changed, the change only has to be made in the DTD.

3 eXtensible Style sheet Language (XSL)

This chapter describes the general concept of XSL and the related topics of XSL: the XSL templates, XSL transformation language (XSLT), the eXtensible Query Language (XQL) and Cascading Style Sheets (CSS).

3.1 General Concepts

XSL is derived directly from the Document Style Semantics and Specification Language (DSSSL), which is the style sheet language for SGML. Some additions have been made to XSL, and these additions will be made to DSSSL as well so the parent/child relationship between them is kept [18]. XSL is a formatting language built especially for use by XML documents. In fact, XSL is an application of XML, i.e. XSL structure and syntax are the same as those of XML. XSL is built around the style sheet mechanism. Style sheets are generally used to apply styles of formatting information consistently throughout a document. The use of XSL makes it possible to transform and present content within an XML file, see figure 11. (Styled Flow Objects means that the XML document can be transformed to almost any document, for example to a pdf-, ps-, word-document, etc.) XSLT is the transformation language for XML. XSLT is a part of the XSL language and XQL is a part of XSLT. XQL makes it possible to search through data and generate XML views of a particular XML file. XSL templates are also a subset of XSLT and are used to match different nodes, see figure 12 for an overview of relations between XML and XSL. However, Cascading Style Sheets (CSS) is not a part of XSL, it is the counterpart of XSL, the competitor. CSS is described here because it does basically the same thing as XSL, it styles XML documents and is quite important since it is used in the HTML world.

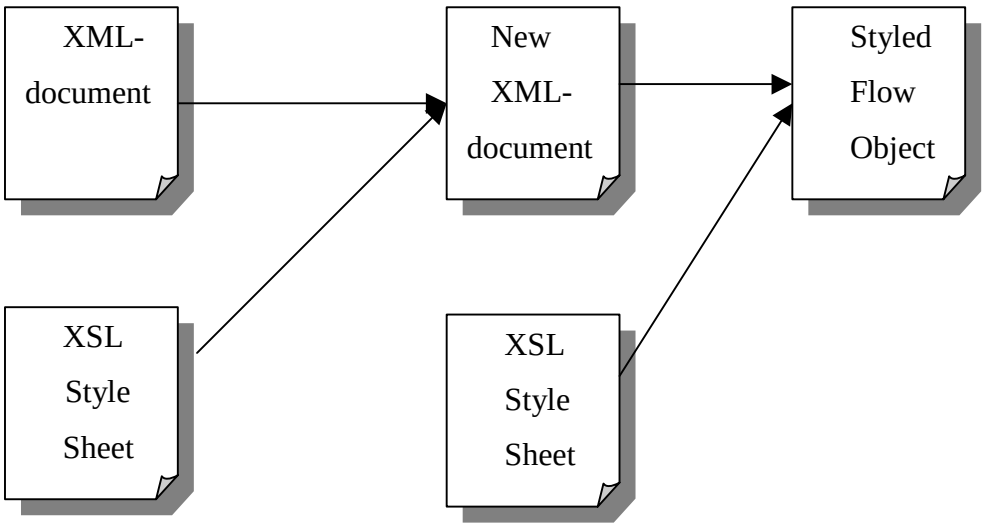

Figure 11. The use of XSL

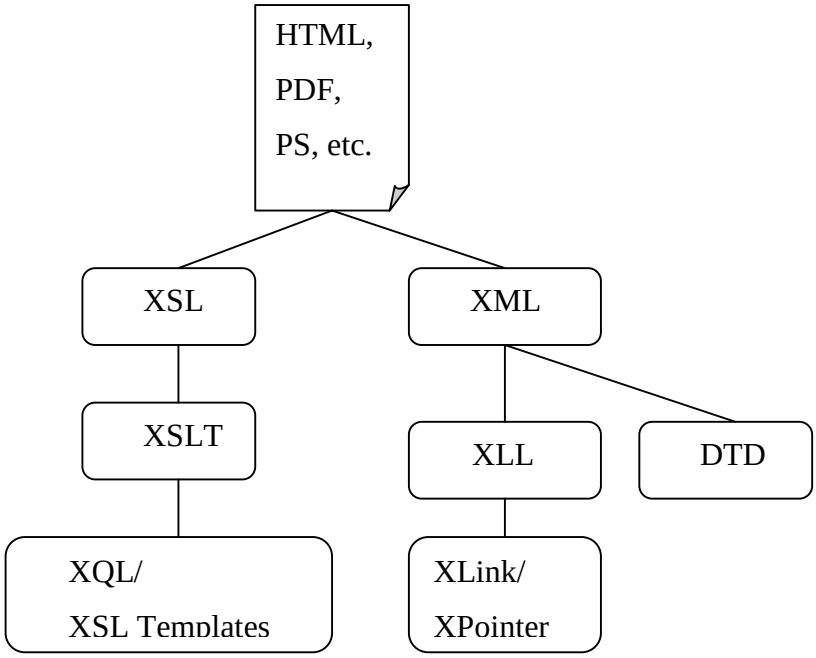

Figure 12. Overview of relations in XML and XSL

3.2 XSL Transformation Language (XSLT)

The XSL transformation language is a description of how a processor can transform an XML document from one format to another. A transformation expressed in XSLT describes rules for transforming a source tree into a result tree. The source tree is tree structure the XML document has before it is styled with XSLT into a result tree. The transformation is achieved by associating patterns with templates. A pattern is matched against elements in the source tree. When constructing the result tree, elements from the source tree can be filtered,

reordered and arbitrary structure can be added. The most obvious use here would be to convert between XML and HTML, although this does not have to be the case, since the transformation process is totally independent of the final output. It allows a style sheet to be applicable to a wide class of documents that have similar source tree structures [20]. This allows a tremendous extensibility in the future, since XSL could transform documents into other formats that have not been thought of yet [14]! In the following, some key features of the XSL language are described [8]:

- The value of a node is a pure text (i.e.., no markup) string containing the contents of the node. This can be calculated by the **"xsl:value-of"** element.
- One can process multiple elements in two ways: By the **"xsl:apply-templates"** element and the **"xsl:for-each"** element.
- The **"xsl:element", "xsl:attribute"**, **"xsl:processing-instruction"**, **"xsl:comment"**, and **"xsl:text"** elements can output elements, attributes, PIs, comments, and text calculated from data in the input document. If one set the **xml:space** attribute to preserve, one can keep all the white space.
- The **"xsl:copy"** element copies the current input node into the output.
- The **"xsl:sort"** element can reorder the input nodes before copying them to the output.
- Whitespace is maintained by default unless an **"xsl:strip-space"** element or **"xml:space"** attribute says otherwise.
- The **"xsl:if**" element produces output if, and only if, its test attribute is true.
- The **"xsl:choose"** element outputs the template of the first **"xsl:when"** element whose test attribute is true, or the template of its **"xsl:default"** element if no **"xsl:when"** element has a true test attribute.
- The **"xsl:import"** and **"xsl:include"** elements merge rules from different style sheets.
- Various attributes of the **"xsl:output"** element allow you to specify the output document's format, XML declaration, document type declaration, indentation, encoding and MIME type.

3.2.1 XSL Templates

XSL is built on the notion of templates. An XSL template provides the mechanism for applying formatting information to data that matches a specific pattern [14]. An XSL template rule is an "xsl:template" element with a match attribute. When a match is found, the contents of the template are transferred to the output. The XSL templates specify a pattern, or

a filter, that can accurately target the style information. The templates contain information about how the target nodes will be displayed. The key to a working XSL style sheet is the way that templates are matched to elements in the XML document. XSL uses recursion to apply the information in the action part of each template, to the matching XML elements. We can specify different templates for each node type by using an appropriate pattern that will match the nodes to its name, which is usually what we want to achieve. By this technique, we can apply different templates to different types of child elements of the root element. By including more apply-templates instructions, we can apply different templates to the different types of child nodes.

3.3 eXtensible Query Language (XQL)

XQL is a query language for XML files. With XQL, one can generate views of the XML document. For instance, one might only want to display special parts of the XML document. The "select" and "match" attributes in an XSL sheet are XQL expressions. All the search possibilities that XQL has are used directly from the XSL Style Sheet [21]. With this tool, we are able to search through an XML document and create a new XML document from the first. This is done by asking questions to the XML document and, in a database manner, gets new XML documents with views that complies to your questions. A tree is generated with XQL rules in an XSL Style Sheet and the result tree is later styled with another XSL style sheet, which in turn results in a Styled Flow Object. Database developers have taken the ability to execute queries on data stores for granted. However, because XML is a young data technology, querying functionality to current date has been very limited. XQL gives developers the querying functionality they are used to in the database world including the following:

- Functional equivalent to the SQL SELECT statement
- Functional equivalent to the SQL WHERE statement
- Boolean logic operation (e.g., AND, OR, NOT)
- Comparison operators (e.g., greater than, less than, less than or equal, or equal to)
- Wildcard operators (e.g., *)

XQL is a notation for retrieving information from a document. The information could be a set of nodes, information about nodes, information about node relationship, or derived values. The specification does not indicate the output format. The result of a query could be a node, a list of nodes, an XML document, array or some other structure.

3.4 Cascading Style Sheet (CSS)

First, it is important to notice that CSS is not related at all to XSL. CSS is the counterpart and was available before XSL. The most common used form of style sheet on the web today is based on the Cascading Style Sheet (CSS) specification. It is used as a style sheet language in HTML, and it allows authors to define classes of styles that can be applied in documents [14]. CSS gives the possibility for several documents to share the same style sheet. The term cascading refers to the fact that more than one style sheet can influence the presentation of a single document simultaneously [7]. CSS is a style sheet mechanism specifically developed to meet the needs of HTML designers. CSS style sheets can be used to set fonts, colours, white space, positioning and background aspects of a document, see figure 13. In addition, it is also possible for several documents to share the same style sheet, which allows designers to maintain consistent presentation within a collection of related documents without having to modify each document separately. CSS is being modified to fit into the XML world.

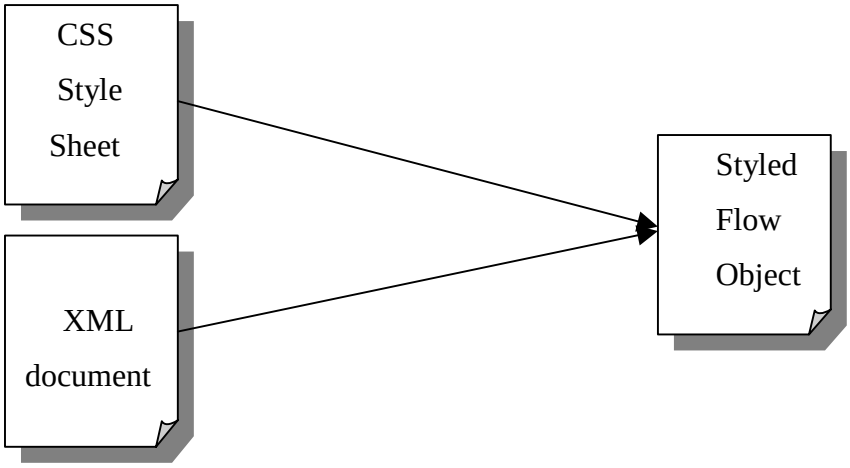

Figure 13. CSS style flow.

CSS2 is the successor of CSS and is a recommendation in the W3C[3] standardisation process. CSS3 is on the Working Draft stage and this gives the signal that CSS probably will be one of the players in the future. Nowadays CSS is not sufficient and that is why W3C is working on XSL as a complement. CSS is probably something that is going to coexist with XSL in some way. CSS will not put XSL experts out of work, but can make the life easier for the HTML people that suddenly have to start style XML documents instead of HTML and this people probably knows CSS already. An overview of the two languages is viewed in figure 14.

[3] World Wide Web Consortium is a group of several companies that sets standards together.

	CSS	**XSL**
Can be used with HTML	Yes	No
Can be used with XML	Yes	Yes
Transformation language	No	Yes
Syntax	CSS	XML

Figure 14. Comparison between XSL and CSS

The unique feature of CSS is that it can be used to style HTML documents. XSL, on the other hand, is able to style and transform XML documents. The two languages can complement each other in the transformation and styling process. For the nearest future, when many web-sites are styled with CSS, XSL is needed to transform data into HTML/CSS documents. Nevertheless, this will only be needed as long as old browsers are around which can not understand XML.

4 eXtensible Linking Language (XLL)

This chapter describes XLink with its simple links, extended links, in-line links and out-of-line links. It will also introduce XPointer with its absolute addressing, relative addressing.

4.1 General Concepts

XLL allows XML tags to refer to other XML documents, as well as to specific elements or data within documents. Two complementary languages comprise XLL: XLink and XPointer. XLink can be used by itself, whereas XLink has to be there when using XPointer [18].

4.2 XML Linking Language (XLink)

XLink specifies a set of attributes in the XLink namespace. By adding these to the element tags, one can refer to other documents. XLink uses an XML vocabulary to define all the pieces necessary to build links into XML documents. XLink defines two kinds of links: simple links and extended links [6].

4.2.1 Simple Links

A Simple link is when the target of a link is a single document or a resource, and the link can only be traversed in one direction. Simple links are no more complicated than regular HTML links. The difference between the simple link and the HTML link is that the simple link is not designated to a specific tag name like "A" in HTML. In XLink, you may call the link whatever you like and that makes it possible to have many different types of linking elements in your document with different behaviour [1].

4.2.2 Extended Links

An extended link, is when the target can be several documents or resources, and the link can define a relationship between those resources. The relationship allows the links to be bi-directional. For an in-line link, the target is part of the local resource of the link. For an out-of-line link, the target is not part of the local resource of the link [18]. One of the capabilities that are required for extended links are applying and filtering sets of relevant links on demand [17]. To define extended links, there is a way of specifying a set of target resources or documents. This is done by using three special values for the reserved “xml:link” attribute:

- “locator” – specifies that this link points to one of the resources or documents within an extended link, see figure 15.
- “group” – indicates that the link is itself a group of document links, and the link element encloses a list of these links see figure 16.
- “document” – indicates that this is a link to a document or resource within a group.

To give a general idea of how extended links looks like see figure 16 [5].

```
<mylink xml:link=”extended” inline=”false”>
  <mytarget xml:link=”locator” inline=”false” role=”Font”
  title=”How to specify different fonts”
  href=http://www.mysite.se/fonts.xml />
  <mytarget xml:link=”locator” inline=”false” role=”Colors”
  title=”How to specify different colors”
  href=http://www.mysite.se/colors.xml />
  <mytarget xml:link=”locator” inline=”false” role=”Text size”
  title=”How to specify different textsizes”
  href=http://www.mysite.se/textsize.xml />
</mylink>
```

Figure 15. Links using locator

```
<mylink xml:link=”group” inline=”false”>
  <mytarget xml:link=”document”
  href=http://www.mysite.se/document1.xml />
  <mytarget xml:link=”document”
  href=http://www.mysite.se/document2.xml />
  <mytarget xml:link=”document”
  href=http://www.mysite.se/document3.xml />
</mylink>
```

Figure 16. Links using group and document

The role attribute specifies what role the target resource plays in the link. The inline=”false” specifies that this is an out-of-line link.

4.3 XML Pointer Language (XPointer)

XPointer is a language that can be used to specify single and multiple locations of a target document, each of which can encompass any section of that document. They can also be used

as the target for XLink elements, to provide a flexible way of linking between documents. A XPointer is made up from a series of location terms, each of which indicates a location. Locations can be absolute, relative to the last location, or a string-match location [4]. Each location term has a keyword such as id, child, ancestor etc. and can have arguments such as an instance number, element type, or attribute [8].

4.3.1 Absolute Addressing

The concept of absolute addressing yields a couple of absolute location terms. A short introduction to each keyword is listed below [17]:

- If an element begins with a root(), the location source is the root element of the resource, this is the default.
- If the XPointer begins with origin(), the location source is the sub-resource from which traversal was initiated, rather than from the default root element. This allows XPointer to select abstract items such as “the next chapter”.
- If an XPointer begins with id(Name). The location source is the element in the resource with an attribute having a declared type of ID and a value matching that given name.
- If an XPointer begins with html(NAMEVALUE), the location source is the element which has an attribute called NAME where the value is the same as the supplied NAMEVALUE. This is the function performed by the “#” fragment identifier in the context of an HTML document.

4.3.2 Relative Addressing

The concept of relative addressing yields a couple of relative location terms. Each of these keywords identifies a sequence of elements or other XML node types from which the resulting location source will be chosen. Below a short introduction to each keyword is given [8]:

- The "child" identifies direct child nodes of the location source.
- The "descendant" identifies nodes appearing anywhere within the content of the location source.
- The "ancestor" identifies element nodes containing the location source.
- The "preceding" identifies nodes that appear before the location source.
- The "following" identifies nodes that appear after the location source.
- The "psibling" identifies sibling nodes that appear before the location source.
- The "fsibling" identifies sibling nodes that appear after the location source.

5 Tools

This chapter describes some of the tools related to XML and used in the prototype application described in chapter 6. The tools are DOM, SAX, XP, and XT.

5.1 General Concepts

It is important to notice that XT, further described in 5.4, is the program that uses all the others; where DOM builds the tree structure, XP is a parser that supports SAX, and SAX reports parsing events via so called callbacks.

5.2 Document Object Model (DOM)

The DOM provides a standard set of objects for representing XML documents, a standard mode of how these objects can be combined, and a standard interface for accessing and manipulating them [16]. DOM establishes a platform and a language neutral interface, which allows programs, or scripts to dynamically access, and update the content, structure and style of XML documents. The W3C designed DOM in such a way that programmers can use it with a variety of object-orientated languages like Java, C++, etc. DOM represents an XML document as a tree, and designates documents, document fragments, document type information, entity references, elements, element attributes, PIs, comments, text, CDATA sections, entities and represents all as nodes. XML is being used as a way of representing different kinds of information that may be stored in different systems, and much of this would traditionally be seen as data rather than as documents. With the Document Object Model, programmers can build documents, navigate their structure, and add, modify, or delete elements and content. All that is found in an XML document could be accessed, changed, deleted, or added using this model [15].

5.3 SAX

SAX is an API for event-based XML parsing [11]. An event-based API reports parsing events (such as the start and end of elements) directly to the application through callbacks, and does not build an internal tree. Tree-based APIs usually puts a great strain on system resources, especially if the documents are large. When using event-based systems, one can parse much

larger documents than the available system memory. Consider, for example, if the user wants to locate just one value and the XML document is very large. Then it would not be very efficient to construct and traverse an in-memory parse tree just to locate this one piece of information. An event-based interface would allow you to find the information in a single pass using a small amount of memory. To understand how an event-based API can work see figure 17 and 18.

```
<?xml version="1.0">
<doc>
  <para>Hello, world!</para>
</doc>
```

Figure 17. First example of event based XML parsing.

An event-based interface will break the structure of this document in a series of consecutive events:

```
start document
start element: doc
start element: para
characters: Hello, world!
end element: para
end element: doc
end document
```

Figure 18. Second example of event based XML parsing

By this method, it is not necessary to cache the entire document in a memory tree.

5.4 XT

XT [10] is a transformation tool written in Java. It is used to transform XML files into other types of files, for example HTML files. It implements the working draft of XSLT.

5.5 XP

XP is an XML 1.0 parser written in Java that supports SAX [9]. Among other things it detects non well-formed documents but is currently not a validating XML processor. XP is designed to be 100% conformant to the XML 1.0 specification.

6 Prototype Application

6.1 Overview

In the Prototype application an XML Document Writer is constructed that reads from a database and tags up the data into an XML format, see figure 19.

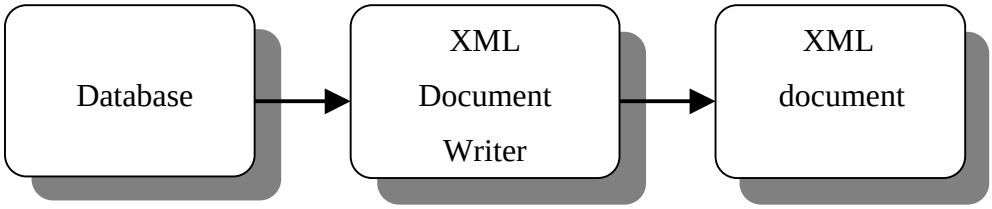

Figure 19. Overview of the prototype application

The next step in the implementation was to take the generated XML document and execute it together with an XSL document in an XSL processor, see figure 20. In this case, we use James Clarks XT together with XP and SAX. Once an XML document is loaded in the parser, it creates a DOM structure. This DOM structure is then styled with XSL. In the prototype application, HTML is generated but it is possible to generate whatever formats is specified, only with different effort and time spent to achieve the other format. In chapter 6.2.1, the XML Document Writer is described.

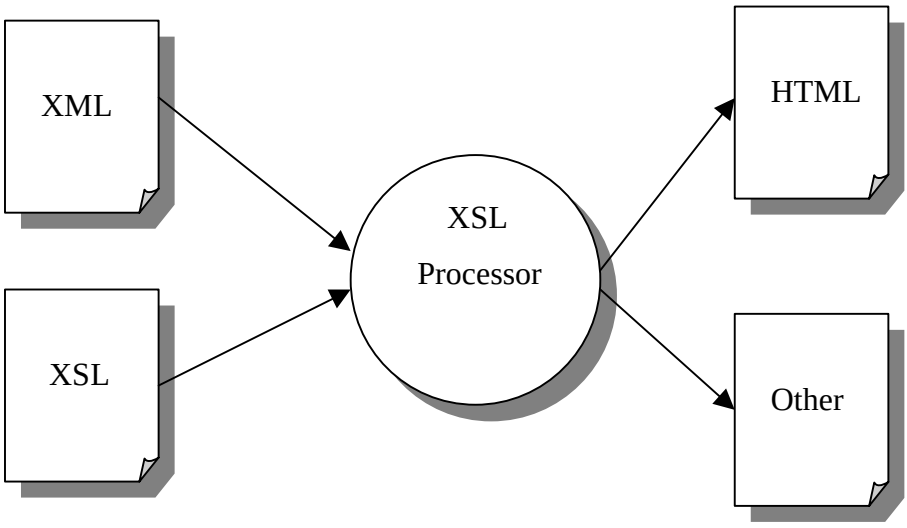

Figure 20. Transforming XML into HTML

6.2 Detailed Description

The implementation of the XML Document Writer is described below. The application takes data from a database and tags up this data into a well-formed XML document. The XML document writer is written in C++ with a procedural design. The program consists of three files: cnaxml_main.cc, cnaxml.cc and cnaxml.hh. The functions are described in chapter 6.2.1. In the implementation, an XSL file was created to style the XML document into an HTML document. This was done because we wanted to put the result on the web using available browsers, which had no support for XML. In chapter 6.2.2, the XSL file is described that was created during the implementation. The transformation tool XT is described in chapter 6.2.3.In chapter 6.2.3, it is also described how it is used, what scripts that is necessary to execute and a picture of the resulting HTML page. Note that Figure 21 is a very shortened version of the generated XML document.

6.2.1 Functions

- cnaxml_main.cc - Here is the database initialized and opened. It accepts arguments from the command line where the first argument defines from which CNA[4] Area network data will be exported. The second argument defines from what domain that data should be exported. The third argument defines from which MO (network object) that data should be exported. The main function then calls the printAll function that appears inside the file cnaxml.cc
- cnaxml.cc - The printAll function prints the XML declaration and the root element <cnaxml>. It then calls the printMOs function where the <mos> tag are printed. Then the program iterates through all MOs before printing the mos end tag. The program calls the printMO function for each MO. In the printMO function, it prints the mo tag, it then prints the identity tags and the domain tag. It checks if there are more mos to print and if so it calls the function printMOs again to make the procedure again. If there are no more MOs at this time, the mo end tag is printed. The program returns to the printAll function and the end tag cnaxml is printed. The document is now a well-formed XML document, see figure 21 for an example.

[4] Cellular Network Administration

```
<?xml version="1.0"?>
<cnaxml>
   <mos>
      <mo>
         <identity>BSC0201</identity>
         <domain>BSC</domain>
         <params>
            <param>
               <name>ACTIVE_OF</name>
               <value>1,2,3,4,5,6,7,8,9,10</value>
            </param>
            <param>
               <name>BSC_NAME</name>
               <value>"BSC0201"</value>
           </param>
           <param>
              <name>BSCMC</name>
              <value>"OFF"</value>
           </param>
           <param>
             <name>GSM_TURGEN</name>
             <value>2</value>
           </param
           <param>
              <name>VERSION</name>
              <value>"AXE,CME20 BSS R7"</value>
           </param>
         </params>
      </mo>
   </mos>
</cnaxml>
```

Figure 21. Example of an XML file

6.2.2 Description of the XSL file

Here is an example of an XSL text that is used to generate the HTML page:

```
<!--XML declaration -->
<?xml version="1.0"?>
<!—The XSL declarations which are like the header files in       -->
<!—the C language.                                              -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/XSL/Transform/1.0"
                xmlns:xt="http://www.jclark.com/xt"
                extension-element-prefixes="xt">
<!—-Start at the root and print the first HTML tags.            -->
<!--The command apply-templates is needed for the recursion     -->
<!--work, one has to apply to the templates one want to match   -->
<!--in the next match block.                                    -->
   <xsl:template match="cnaml">
      <html>
        <head>
          <title>Object Structure</title>
        </head>
        <body bgcolor="ffff00">
             <P><font face="Times, bold" color="black" size="5">Object
             Structure</font></P>
        </body>
        <xsl:apply-templates select="//mo"/>
      </html>
    </xsl:template>
```

```
<!—- Here we match each cell and prints a indent figure          -->
<!-- depending which cell is going to be processed.              -->
     <xsl:template match="mo">
      <xsl:choose>
        <xsl:when test="domain='BSC'"></xsl:when>
        <xsl:when test="domain='Internal Cell'"><xsl:text>|-
           </xsl:text></xsl:when>
        <xsl:when test="domain='Neighbour Relation'">|--</xsl:when>
        <xsl:when test="domain='Channel Group'">|--</xsl:when>
           <xsl:when test="domain='External Cell'">|-</xsl:when>
        <xsl:when test="domain='Priority Profile'">|-</xsl:when>
           <xsl:when test="domain='Transceiver Group'">|-</xsl:when>
           <xsl:when test="domain='Site'">|-</xsl:when>
           <xsl:when test="domain='Overlaid Subcell'">|--</xsl:when>
      </xsl:choose>
<!—-Here is hyperlinks created. They appear in the right frame.  -->
<!--xt.document is the command that creates new html pages       -->
<!—- with all information about each cell.                       -->
      <a href="{concat(identity,'.htm')}" TARGET="hoger">
          <xsl:value-of select="identity"/></a><br></br>
      <xt:document method="html" href="{concat(identity,'.htm')}">
        <html>
          <head>
            <title><xsl:value-of select="identity"/></title>
          </head>
             <body bgcolor="ffff00">
<!—Prints the first tableheaders.                               -->
          <table BORDER="3" CELLSPACING="1" CELLPADDING="1">
            <th>Identity</th>
            <th>Domain</th>
               <p>
<!—Prints the identity and domain names.                        -->
                <tr>
               <td>
                 <font face="Times, bold" color="purple" size="3">
                      <xsl:value-of select="identity"/>
                 </font>
               </td>
               <td>
                 <font face="Times, bold" color="purple" size="3">
                      <xsl:value-of select="domain"/>
                 </font>
               </td>
             </tr>
            </p>
          </table>
          <xsl:apply-templates select="params"/>
           </body>
        </html>
      </xt:document>
    </xsl:template>
<!—Here the headers of the  second table is printed out.         -->
    <xsl:template match="params">
      <table BORDER="3" CELLSPACING="1" CELLPADDING="1">
        <th>Parameter</th>
        <th>Value</th>
        <xsl:apply-templates select="param" />
      </table>
    </xsl:template>
<!—Here the HTML document is generated.                         -->
    <xsl:template match="param">
      <tr>
```

```
        <td>
             <font face="Times, serif" color="blue" size="2">
               <xsl:value-of select="name"/>
             </font>
        </td>
        <td>
           <font face="Times, serif" color="green" size="2">
              <xsl:value-of select="value"/>
           </font>
           </td>
      </tr>
    </xsl:template>
</xsl:stylesheet>
```

6.2.3 Description of the Transformation Tool

In the implementation, XT is used to transform the XML and XSL file to a number of HTML files. A script, see figure 22, is used as a makefile to compile the XML and XSL file into HTML files. On the command line, the user gives three arguments. First the name of the XML file, then the XSL file, and last the name of the HTML file, which we are going to generate, see figure 23. The user is bound to have the third argument intact as lx.htm so that all the hyperlinks are generated to that particular file. In the HTML document, frames are used, see figure 24. When the user activates one of the hyperlinks in the left frame, the right frame which is called rx.htm, shows the content of each generated HTML file, see figure 25.

```
#!/bin/sh

if [ ! $# = 3 ]
then
   echo "Usage: xslt source stylesheet result"
   exit 1
fi

CLASSPATH=<xtpath>/xt.jar:<xtpath>/xp.jar:
<xtpath>/sax.jar:$CLASSPATH

export CLASSPATH

java com.jclark.xsl.sax.Driver $1 $2 $3
```

Figure 22. The script “xslt” for running XT

```
xslt  xmlfile.xml xslview1.xsl lx.htm
```

Figure 23. Command line arguments

```
<html>
   <head>
      <title>CNAML</title>
   </head>
   <frameset cols="20%, 80%">
      <frame name="vanster"
      src="lx.htm"
      scrolling="yes">
      <frame name="hoger"
      src="rx.htm"
      scrolling="auto">
   </frameset>
</html>
```

Figure 24. Example of the index.html file

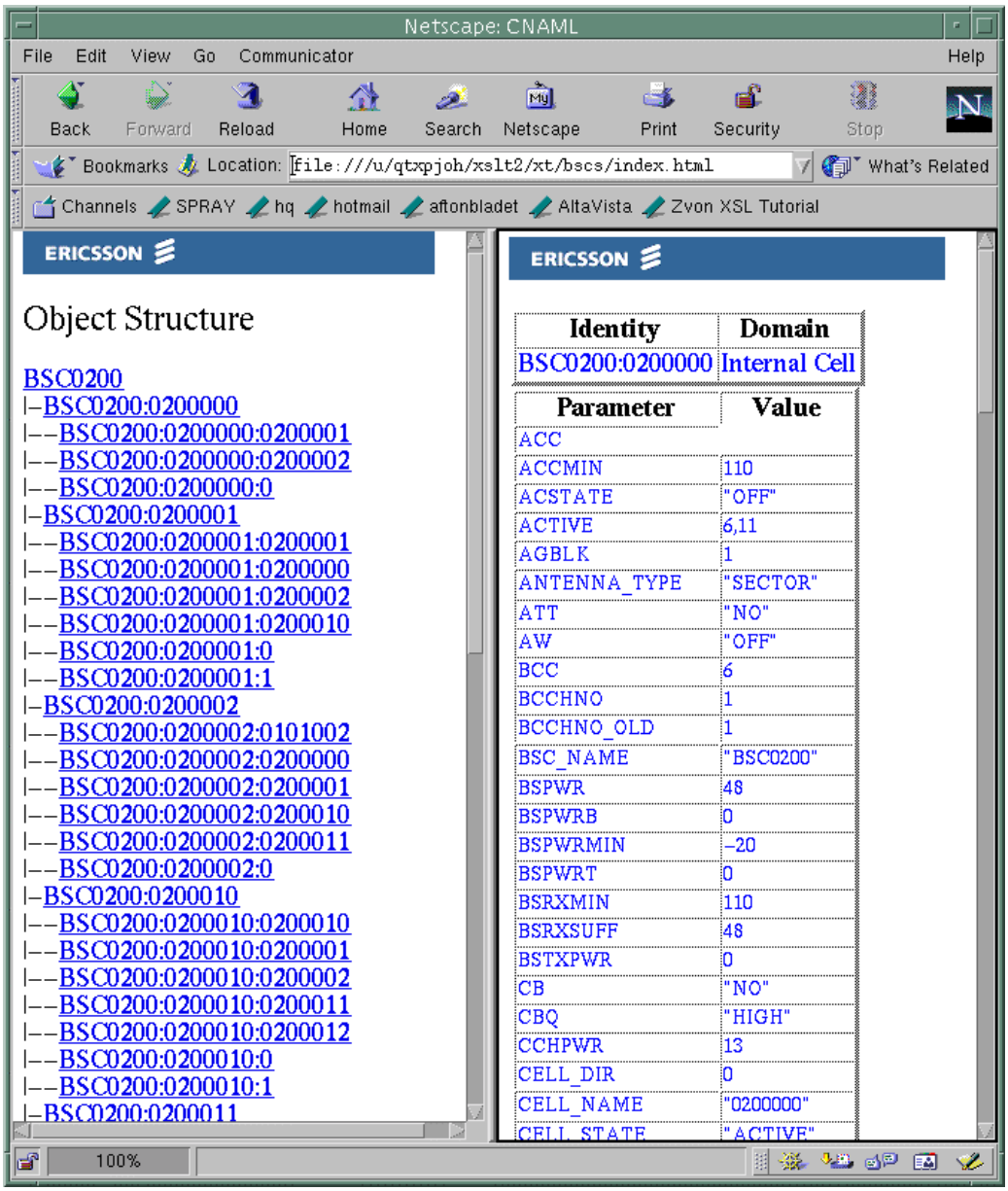

Figure 25. Example of a generated HTML page

7 Conclusion

XML is the format that forthcoming web-applications probably will be based on. XML complements HTML and makes the web-development more powerful, with more new features coming all the time. It is a constant field of development, i.e., new features and standards emerge all the time. Many large vendors such as Microsoft, Oracle, IBM, etc., have XML-products on the market already. The great thing with XML is that one can define own tags. This feature makes the language extensible. The vision of XML is that one can transfer all different kinds of formats over the Internet and all applications should understand each other. That is probably just a vision but some of them will be able to understand each other. The goal is difficult to achieve because all different vendors have to use the same DTD, and that seems to be a difficult goal to achieve.

Considering the role of XSL in the future, this probably will be the language for both transforming and for styling. Now CSS is developed with version 2, and version 3 is also on the way. The people behind the XSL-specification have the minimum goal that XSL shall have all the functionality that both CSS and DSSSL has. The fact that XSL has XML syntax makes it even easier to believe that XSL is the language for the future. In the future, XSL will probably take over because it is better to transform and style using the same language. It is also possible to separate the data from all the styling that is not possible in CSS. That is why XSL probably will win in the end.

A problem during the implementation has been that XSL is still under development. It has changed radically in the past and will change in the future. All products available now implement different subsets of the current draft specification. Furthermore, many products including Internet Explorer 5.0 and XT add elements not actually present in the current draft specification for XSL. Consequently, very few examples will work exactly the same in different software. This is a problem for developers and hopefully these problems will be smaller in the future when the specification has turned into a standard for XSL.

References

[1] Charles F. Goldfarb. Paul Prescod. The XML Handbook. Prentice-Hall, 1998.

[2] Ian S. Graham. Liam Quin. XML Specification Guide. Wiley, 1999.

[3] Elliotte Rusty Harold. XML Bible. IDG Books Worldwide, 1999.

[4] Steven Holzner. XML complete. McGraw-Hill, 1998.

[5] Alex Homer. XML IE5. Wrox, 1999.

[6] William J. Pardi. XML in action Web Technology. Microsoft press, 1999.

[7] John E. Simpson. Just XML. Prentice Hall, 1999.

[8] http://metalab.unc.edu/xml/books/bible/updates/14.html.

[9] http://www.jclark.com/xml/xp/index.html.

[10] http://www.jclark.com/xml/xt/index.html.

[11] http://www.megginson.com/SAX/index.html.

[12] http://www.mozilla.org.

[13] http://www.nwalsh.com/docs/articles/xml/index.html.

[14] http://www.xml.com/xml/pub/r/DOM_Level_1.

[15] http://www.w3.org/TR/1998/PR-DOM-Level-1-19980818/introduction.html.

[16] http://www.w3.org/TR/NOTE-XSL-and-CSS.

[17] http://www.w3.org/TR/WD-xptr.

[18] http://www.w3.org/TR/WD-xlink.

[19] http://www.w3.org/TR/WD-xsl.

[20] http://www.w3.org/TR/WD-xslt.

[21] Peter Rosengren.Webbens universalformat tar fart. *Datateknik*, Nummer 7, sid 28-30, 1999.